ADMINISTRACIÓN DE SERVICIOS DE INTERNET

De la teoría a la práctica

F. Maciá / F. J. Mora / J. A. Gil / V. Gilart
D. Marcos / J. V. Berná / J. C. Monllor
H. Ramos / A. Albaladejo / A. Hernández

TD
TEXTOSDOCENTES

PUBLICACIONES
UNIVERSIDAD DE ALICANTE

ADMINISTRACIÓN DE SERVICIOS DE INTERNET

De la teoría a la práctica

MACIÁ PÉREZ, MORA GIMENO,
GIL MARTÍNEZ-ABARCA, GILART IGLESIAS,
MARCOS JORQUERA, BERNÁ MARTÍNEZ,
MONLLOR PÉREZ, RAMOS MORILLO,
ALBADALEJO BLÁZQUEZ Y HERNÁNDEZ SÁEZ

ADMINISTRACIÓN DE SERVICIOS DE INTERNET

DE LA TEORÍA A LA PRÁCTICA

PUBLICACIONES DE LA UNIVERSIDAD DE ALICANTE

Publicaciones de la Universidad de Alicante
Campus de San Vicente s/n
03690 San Vicente del Raspeig
Publicaciones@ua.es
http://publicaciones.ua.es
Teléfono: 965903480
Fax: 965909445

ISBN: 978-84-7908-989-4
Depósito legal: MU-2302-2008

Diseño de portada: candela ink.
Impresión: Compobell, S.L.
C/. Palma de Mallorca, 4 - bajo
30009 Murcia

A la memoria de nuestro querido amigo Alfonso

ÍNDICE

PRÓLOGO

¡Ojalá que vivas tiempos interesantes!
antigua maldición china…

... Y es que, en realidad, esta antigua maldición china hace referencia a la imposibilidad de desvincular la renovación, los nuevos puntos de mira y la esperanza que los *tiempos interesantes* conllevan, de sus, en muchas ocasiones, terribles consecuencias, escenarios y situaciones: guerras, revoluciones o profundos cambios sociales.

Esta es la situación que nos está tocando vivir dentro del ámbito de las tecnologías de la información y de las comunicaciones: *tiempos interesantes para las TIC*. Cada día aparece una nueva tecnología que promete ser mejor que las existentes, que revolucionará el mercado y que, en realidad, sin que ni tan siquiera haya alcanzado un mínimo de estabilidad en su curva de madurez, desaparece sin haber dejado apenas rastro.

Nuestra particular *revolución* se centra en ser capaces de establecer un equilibrio entre estar permanentemente actualizados y saber distinguir oportunamente cuáles serán las tecnologías más apropiadas en la actualidad y que nos permitan construir sistemas y servicios lo más duraderos posible.

Cada vez más, lo realmente importante es el propio usuario y sus necesidades, de forma que no tenga que conocer ni ocuparse de los detalles de la infraestructura tecnológica subyacente. Sin embargo, a medida que la tecnología se vuelve invisible a sus ojos, tanto más complejos, ricos y diversos resultan los niveles tecnológicos más bajos responsables de proporcionarles soporte.

Toda esta complejidad, riqueza y diversidad la podemos trasladar directamente al ámbito de la gestión de sistemas, recayendo sobre sus administradores gran parte de la responsabilidad de su funcionamiento continuado.

Si analizamos cuáles pueden ser los factores de éxito que hagan viable la oferta de servicios a ciudadanos y organizaciones, a través de las nuevas tecnologías, a precios asequibles, tiempos asumibles e infraestructuras sostenibles y confiables, no los encontraríamos tanto en el despliegue de *súper-sistemas* desarrollados ad hoc, como en la capacidad de integración de diferentes tecnologías, desarrolladas y mantenidas por terceros, de forma que —alineadas bajo unos mismos objetivos, respetando las mismas políticas y siguiendo estrategias comunes— proporcionen los servicios deseados.

Es por esta razón que el principal objetivo de este documento se centrará en exponer, aclarar y organizar los diferentes conceptos, tecnologías, modelos y herramientas involucrados en el ámbito del desarrollo de sistemas distribuidos, de forma que podamos conseguir una visión global y coherente de las TIC.

El libro está organizado en dos grandes secciones:

- *Parte I. Servicios de Red*
- *Parte II. Servicios avanzados para la Red.*

La primera parte, Servicios de Red, está dedicada a aspectos que hoy en día se pueden considerar generales para entender los conceptos y tecnologías asociados con la implantación de aplicaciones distribuidas. Los casos prácticos recogidos en cada capítulo de esta sección están más orientados a administradores de sistemas. Los capítulos que se incluyen en esta sección son:

- TECNOLOGÍAS WEB BÁSICAS
- AMPLIACIONES EN EL CLIENTE
- AMPLIACIONES EN EL SERVIDOR
- SERVIDOR DE APLICACIONES
- COPIAS DE SEGURIDAD
- CORREO ELECTRÓNICO

La segunda parte, Servicios avanzados para la Red, está dedicada a servicios de carácter general, que suelen propocionar soporte, no a los usuarios finales, sino a otras aplicaciones y servicios de red. Debido a la

complejidad de estos servicios y a su relación, incluso dependencia, con los servicios básicos, se ha estimado que era preferible otorgarles un tratamiento particularizado. Los servicios recogidos en esta parte son:

- SEGURIDAD
- SISTEMA DE ARCHIVOS DISTRIBUIDO
- SERVICIOS DE DIRECTORIO
- ACCESO A SERVICIOS MEDIANTE LDAP

A su vez, cada capítulo ha sido diseñado para que se pueda ser autocontenido y que el lector pueda realizar una lectura independiente según sus intereses. Según esta idea, se le ha dotado de una estructura en la que se expone en primer lugar los principios teóricos y prácticos de cada tema para pasar a un segundo apartado en el que se concluye cada capítulo con un caso de uso basado en alguna tecnología concreta.

Todo esto hace que el perfil del profesional al que va dirigido el documento sea bastante amplio. Si el lector tan sólo desea tener una noción general de las diferentes tecnologías existentes alrededor del desarrollo aplicaciones web distribuidas, bastará con centrarse en la primera parte del libro y, dentro de cada capítulo, saltarse los casos de uso. Si el lector es un administrador de sistemas más o menos avanzado y lo que desea es conocer cómo funcionan unos servicios concretos, puede realizar una lectura radicalmente diferente concentrándose sólo en los diferentes casos de uso que se han incorporado al final de todos los capítulos del libro, Por supuesto, entre ambos perfiles, cabe cualquier combinación en función de los conocimientos y de los intereses de cada lector.

PARTE I
SERVICIOS DE RED

1. TECNOLOGÍAS WEB BÁSICAS

En sus orígenes, la tecnología Web, basada en el protocolo HTTP para la transmisión de hipertexto sobre protocolos de comunicaciones TCP/IP, sirvió para proporcionar una nueva visión de Internet. Si antes, con servicios como FTP, la Red se concebía como un conjunto de nodos servidores cuya topología y estructura física había que conocer, ahora se logra un nivel de abstracción mayor: lo que importa es la información —*los contenidos* en términos del WWW— en lugar de su ubicación, de las características de los servidores en los que está ubicada o de los dispositivos empleados para acceder a la misma: computador personal, PDA, WebTV, teléfono móvil, etc.

Estas características de independencia entre clientes y servidores, unida al auge de la generación de contenidos en Internet y la enorme heterogeneidad reinante en el entorno de las TIC, facilitan que prospere rápidamente este nuevo modelo cliente/servidor.

Sin embargo, debido en parte a su paulatina participación en el mundo de los negocios y a la necesidad, por tanto, de generación de contenidos dinámicos, cada vez son más las tecnologías que en los últimos tiempos han venido a acompañar a los tradicionales *servidores Web* y, porqué no, a *engordar* nuestros *navegadores Web*. Está claro que se necesita una mejor gestión de recursos y una mayor organización de las tecnologías implicadas.

Atrás quedaron los tiempos en que la Web era un mero escaparate de consulta de información estática, dónde las empresas presentaban su catálogo de productos o su oferta de servicios. En la actualidad, se tiende progresivamente a hacer efectivos los diferentes servicios que ya se prestan por otros medios, en esa red universal que es Internet. Para ello, las

aplicaciones convencionales cliente/servidor para redes locales o corporativas, escritas pensando en un entorno determinado, deberán ser adaptadas a las nuevas características e idiosincrasia que imponen las tecnologías de Internet en general y de la Web en particular.

Durante los próximos cuatro capítulos revisaremos su evolución: partiendo en este capítulo del concepto de HTTP básico y de aplicación Web, en los capítulos 2 y 3 analizaremos las tecnologías más importantes que se han ido incorporando tanto en la parte del cliente —DTML, *ActiveX*, *applets java*, *plug-ins*— como en la del servidor —*páginas activas*, componentes en el servidor, contenedores de componentes; finalmente, en el capítulo 4 veremos cómo, de alguna manera, tanto por la complejidad que están adquiriendo las actuales aplicaciones como por la sobrecarga que todas estas tecnologías infligen sobre los servidores Web convencionales, en la actualidad, la tendencia es volver a aligerar a nuestro servidor web, dejándolo que se ocupe de aquello para lo que fue concebido —la gestión del nivel de presentación o interfaz con el usuario final— y organizar un sistema paralelo complementario capaz de dar soporte adecuado a las aplicaciones.

1.1. MODELO CLIENTE-SERVIDOR

El Modelo Cliente-Servidor es uno de los más empleados por un equipo o aplicación (cliente) para acceder a recursos ubicados en otro equipo (servidor) u ofrecido por otra aplicación (servicio).

En la figura 1.1 se muestra un diagrama en el que se destacan los principales elementos que intervienen en un entorno preparado para emplear el modelo cliente-servidor a través de Internet.

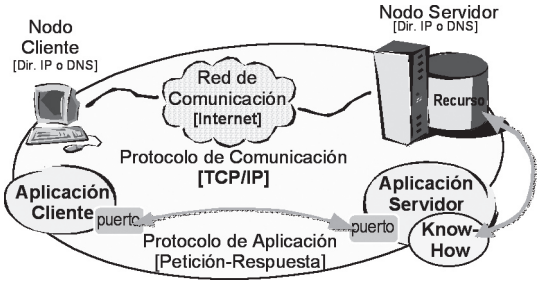


Figura 1.1. Principales elementos del modelo Cliente/Servidor sobre TCP/IP.

El primer elemento a destacar es el *nodo cliente*. El nodo cliente está formado por el equipo computacional —ya sea un PC convencional, un portátil, una agenda electrónica o un teléfono móvil— con su respectivo sistema operativo y con capacidad para conectarse a una red de computadores. Cada nodo cliente debe disponer de, al menos, una aplicación que denominaremos *aplicación cliente*. Esta aplicación es la responsable de solicitar el recurso o servicio deseado y, en caso de ser interactiva, actuar como interfaz con el usuario final. En la práctica, tanto el nodo cliente como la aplicación cliente se suelen denominar, sencillamente, clientes.

Al otro lado del escenario nos encontramos el *nodo servidor* o *servidor*. En este caso, se trata del equipo —la combinación de hardware más sistema operativo— que posee el *recurso* hardware o software objeto del servicio. El servidor deberá estar conectado de alguna forma a la misma red que el cliente. Sobre él se ejecutará la *aplicación servidora* o *servicio* capaz de atender las solicitudes del cliente y mediante su *know-how* —su *saber cómo*— para acceder y gestionar dicho recurso.

Entre ambos nodos encontramos la *red de comunicación*. Sea cual sea su tecnología física, deberá emplear como protocolo de comunicación (niveles de red y transporte), la *pila TCP/IP*, es decir, el conjunto de protocolos empleado por *Internet*.

En este modelo cliente-servidor resulta imprescindible que ambas partes tengan alguna forma de referenciarse. Un cliente debe poder invocar a un servidor, y este servidor tendrá que ser capaz de devolverle su solicitud. Puesto que la red (lógica) en la que nos basamos es Internet, el mecanismo de identificación que cada nodo conectado a la misma debe poseer —ya sea cliente o servidor—, es lo que denominamos una *dirección IP*. Sin embargo, una dirección IP está compuesta por un número de cuatro bytes que resulta muy incómodo de recordar y manipular por los usuarios humanos. Por esta razón, lo más común es emplear un sistema de traducción de direcciones IP a un sistema de nombres jerárquicos denominando DNS (*Domain Name Server*), en el que el nombre del nodo se expresa mediante una combinación de *nombre* más el *dominio* al que pertenece. La dirección así expresada se denomina *nombre DNS*.

Aunque una dirección IP o un nombre DNS permiten identificar el nodo de red de manera unívoca, todavía falta identificar la aplicación concreta, de entre todas las que se están ejecutando sobre cada nodo, con la que nos queremos comunicar. En este caso, el protocolo TCP/IP proporciona como mecanismo un *número entero* diferente a cada aplicación en

ejecución que desee acceder a la red. Este número se denomina *puerto*. Por lo tanto, el par `direcciónIP:puerto` o `nombreDNS:puerto` será el que realmente identifique cliente y servicio.

Finalmente, aunque los nodos de red cliente y servidor se comunican mediante el protocolo de comunicaciones TCP/IP que les permite entenderse a nivel de red y de transporte de datos, las aplicaciones cliente y servidor también deben poseer un mecanismo que les permita hablar entre ellas, es lo que denominamos *protocolo de aplicación*.

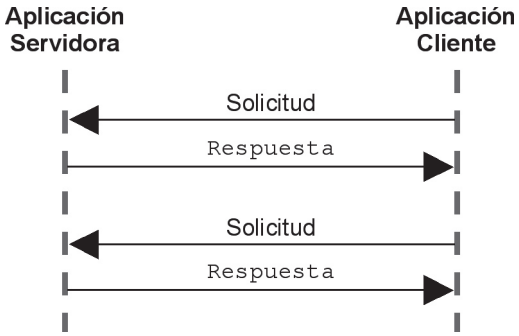


Figura 1.2. Dos secuencias Solicitud-Respuesta genéricas.

En el caso de este tipo de aplicaciones, este protocolo suele ser del *tipo petición-respuesta* (ver figura 1.2). Son protocolos en los que la *respuesta* del servidor a la *solicitud* del cliente se entiende como el justificante de haber sido recibida. Esto permite disminuir el tráfico de red provocado por el envío de justificantes y que, en muchas ocasiones, son el motivo del colapso de los sistemas de comunicación. En la mayor parte de los casos, el protocolo de aplicación es el que proporciona su nombre al servicio: FTP, HTTP, *Telnet*, etc.

1.2. SERVICIO HTTP

El servicio HTTP es un servicio basado en el modelo cliente-servidor sobre Internet, donde el cliente es un *navegador Web* y el servidor es un *servidor Web* (ver figura 1.3), utilizando ambos para su entendimiento el *protocolo de aplicación HTTP* (*HyperText Transfer Protocol*). En este

caso el servidor Web se configura por defecto para escuchar solicitudes en el *puerto 80*, de forma que cualquier cliente pueda encontrarlo fácilmente. El recurso que se ofrece lo constituyen documentos de texto denominados *páginas HTML* (*HyperText Markup Language*), por ser éste el lenguaje que se emplea para su codificación (ver figura 1.3).

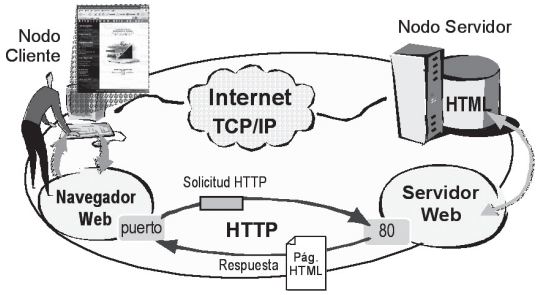


Figura 1.3. Elementos del servicio HTTP.

En la figura 1.4 se puede apreciar un ejemplo de solicitud de página HTML utilizando el protocolo HTTP. La respuesta será la propia página solicitada.

Método	Recurso	Protocolo	Cabecera
GET	/index.html	HTTP/1.1	

Figura 1.4. Ejemplo de una *Solicitud* HTTP.

1.2.1. Protocolo de aplicación HTTP

Se trata de un protocolo de aplicación, basado en texto y del tipo solicitud-respuesta. El formato de una solicitud HTTP responde a la siguiente sintaxis:

```
<Método HTTP> <URI> <Protocolo>
<Cabecera>
<Línea en blanco>
[<Cuerpo>]
```

Donde:

- El *Método HTTP* indica la acción que se solicita al servidor. Alguno de los principales métodos HTTP son los siguientes:
  - `HEAD` Solicita la cabecera del recurso referenciado.
  - `GET` Solicita un recurso mediante una URI.
  - `POST` Solicita un recurso y pasa información adicional en el *cuerpo* de la solicitud.
  - `PUT` Solicita que el servidor almacene la información enviada con el nombre indicado en la URI
- La *URI* (*Uniform Resource Identifier*) es la forma normalizada de referenciar un recurso. La diferencia con una URL (*Universal Resource Locator*) es que en el último caso, además de la URI, también se incluye el tipo de servicio al que se quiere acceder, la dirección del nodo servidor y el puerto en el que el servicio espera las solicitudes.
- El *Protocolo* indica el nombre y versión del protocolo que espera el cliente.
- La *Cabecera* de la solicitud sirve para indicar diferentes parámetros que modifican la forma en la que se realiza la solicitud o la respuesta. La cabecera finaliza con una línea en blanco. El formato de cada parámetro es el siguiente:

  ```
  <etiqueta>: <valor>\r\n
  ```

- El *Cuerpo* de la solicitud es opcional y permite enviar información adicional junto a la solicitud.

En la figura 1.5 se presenta un sencillo ejemplo en el que se solicita al servidor el archivo `miAplicacion.cgi` situado en la carpeta `/cgi`, también se le indica mediante la cabecera algunas condiciones —como que se acepta cualquier tipo de datos, que no cierre la sesión inmediatamente y que nuestro navegador es del tipo genérico—. Además, se emplea el cuerpo de la solicitud para pasar información adicional (que se detallará en los próximos apartados) al servidor Web.

Así mismo, la Respuesta también sigue un formato establecido que se puede sintetizar en la siguiente sintaxis:

```
<Línea de estado>
<Cabecera>
<Línea en blanco>
[<Cuerpo>]
```

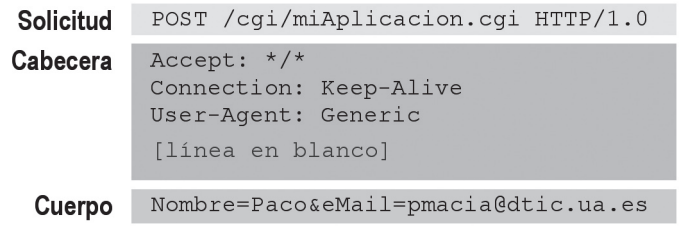


Figura 1.5. Ejemplo de solicitud HTTP.

Donde:

- La *Línea de estado* comienza con un símbolo más (+) si todo ha ido bien o con un símbolo menos (-) si se ha producido algún incidente. A continuación se especifica el protocolo que emplea el servidor (por ejemplo: HTTP/1.0), el código de estado informando sobre el desarrollo, negativo o positivo, de la solicitud y, finalmente, una descripción textual del mismo: si ha sido exitosa, si se ha producido un error o, sencillamente, con información de carácter más general.

```
+|-protocolo código texto
```

- La *Cabecera* de la respuesta sirve para indicar diferente información sobre el servidor y sobre el objeto solicitado: la fecha, el nombre del servidor, el tipo MIME del objeto, etc. El formato de cada parámetro es el siguiente:

```
<etiqueta>: <valor>
```

- El *Cuerpo* es la parte más importante de la respuesta, pues está formado por el contenido del recurso web solicitado, generalmente una página en formato HTML o un archivo en formato MIME (ambos se describirán más adelante).

En la figura 1.6 se presenta un ejemplo de conexión remota mediante un sencillo cliente *telnet*. En primer lugar se realiza una conexión indicando el nombre DNS del servidor y el puerto en el que se encuentra escuchando (puerto 80). Una vez establecida la conexión, el servidor Web queda a la espera de una solicitud que siga el protocolo HTTP. En el caso del ejemplo se trata de una solicitud para recuperar el archivo `index.html` que se debe encontrar en el directorio raíz del servidor Web. El servidor Web responde

indicando con un símbolo más (+) que todo ha ido bien, informa sobre el protocolo con el que trabaja (HTTP/1.1), el código de estado (200) y una breve descripción que indica que todo ha ido bien (OK). A continuación envía la cabecera de la respuesta con algunos datos sobre la conexión y la información a transmitir (fecha, servidor, última modificación del archivo, una marca, el tamaño de los datos y del archivo, el estado de la conexión y el tipo MIME del archivo). Finalmente, tras una línea en blanco que señala el final de la cabecera, utiliza el cuerpo de la respuesta para transmitir el contenido del archivo de texto, en formato HTML, que se le había solicitado. Una vez enviada esta información y puesto que HTTP es un protocolo de tipo petición-respuesta, el servidor da por finalizada la conexión, es decir, si deseamos un nuevo recurso de este servidor, tendremos que realizar una nueva conexión siguiendo nuevamente todos los pasos anteriormente detallados. Por supuesto, un cliente telnet no sabe interpretar las etiquetas de formato que ha recibido, por lo que mostrará el contenido del archivo de forma literal.

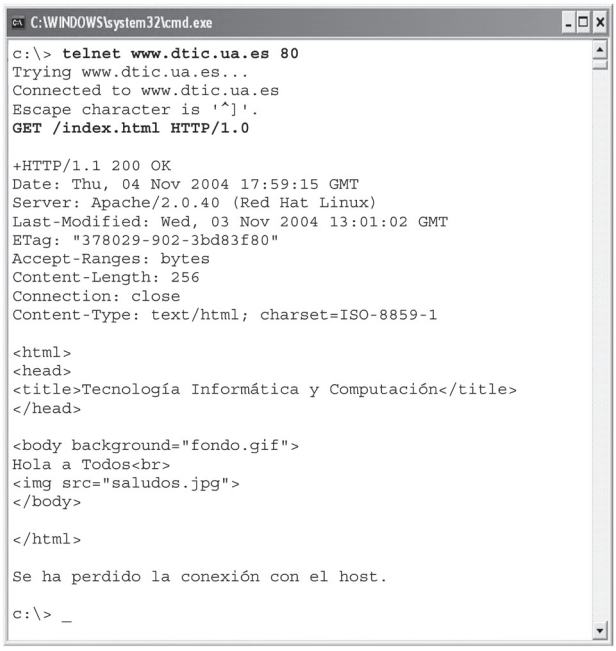


Figura 1.6. Ejemplo de una sesión remota con una solicitud-respuesta HTTP.

En principio, un servidor Web *elemental* sólo está preparado para servir información estática, normalmente en forma de páginas HTML que el navegador Web del cliente se encargará de interpretar y representar, generalmente, de forma gráfica en su monitor. En el ejemplo de la figura 1.6 se había solicitado una sencilla página HTML que implementa el típico caso de ejemplo «*Saludos*». El contenido del archivo `/index.html` con dicho contenido podría ser el mostrado en la figura 1.7.

```
<HTML>
    <HEAD>
        <TITLE>Saludos</TITLE>
    </HEAD>

    <BODY background="fondo.gif">
        Hola a todos <BR>
        <IMG src="saludos.jpg">
    </BODY>
</HTML>
```

Figura 1.7. Listado del archivo `index.html` con un sencillo documento HTML.

Un documento HTML, tal y como sus siglas indican, está formado por los siguientes elementos:

- Un documento de texto.
- Etiquetas de marcado o de formato:

  ```
  <etiqueta [arg1 arg2 ...]> ... [</etiqueta>]
  ```

- Referencias a otros recursos:

  ```
  <A HREF="URL"> texto descriptivo </A>
  ```

Aunque las páginas o archivos en formato HTML son los recursos más básicos de un servicio Web, pueden requerir otros recursos adicionales para que el navegador Web pueda mostrar todo su contenido (en el ejemplo de la figura 1.7, tras el mensaje «*Hola a todos*», se presentará una imagen en formato *JPEG* (*Joint Photographic Experts Group*), ubicada en el servidor, con el nombre de `saludos.jpg`. Por supuesto, el navegador Web tendrá que recuperar este archivo mediante una nueva solicitud al servidor Web. El problema es que estos recursos suelen ser archivos gráficos, de

vídeo o de sonido, que no se encuentran en formato de texto, que es para lo único que está preparado el protocolo de texto HTTP. Para subsanar esta deficiencia, cada archivo binario deberá ser convertido previamente a un archivo de texto mediante un mecanismo de representación externa de datos denominado *MIME* (*Multipurpose Internet Mail Extension*). La figura 1.8 muestra el contenido de un archivo MIME correspondiente a la conversión a texto de una imagen binaria en formato JPEG. Este mecanismo fue originalmente concebido para poder adjuntar archivos en formato binario a los mensajes de correo electrónico mediante el servicio SMTE (que se estudiará en el capítulo 6).

1.2.2. Arquitectura HTTP

Podemos considerar que la arquitectura técnica, física, de un sistema distribuido para la prestación del servicio HTTP tiene como fundamento una red de comunicaciones. Sobre ésta, (a la derecha en la figura 1.9), encontramos la estructura del cliente. Subiendo por las diferentes capas de dicha estructura hallamos el hardware del nodo cliente en el que destaca la tarjeta adaptadora de red (NIC) que le permite acceder al canal de comunicación. A continuación encontramos el sistema operativo que gestiona el nodo y en el que se ubica la pila TCP/IP que implementa los protocolos de comunicaciones. Finalmente llegamos a la aplicación cliente que, en este caso, es un navegador Web. Como ya se ha podido estudiar en el apartado anterior, el navegador Web tendrá capacidad para comunicarse con el servidor mediante el protocolo HTTP y para interpretar los archivos binarios codificados en MIME. Una vez recibida la información que ha solicitado, deberá ser capaz de interpretarla y mostrarla, generalmente de forma gráfica, al usuario final.

Si seguimos analizando esta arquitectura técnica podemos observar la estructura del servidor Web que, desde el punto de vista de los elementos que la conforman y de su organización, no difiere en gran medida de la estructura del navegador web. Como se puede apreciar en la figura 1.9, el servidor está compuesto por un hardware conectado a la red mediante su NIC, un sistema operativo con su propia pila TCP/IP —quizá todo ello mucho más optimizado para soportar mayor carga en las transacciones de red y en el acceso al sistema de almacenamiento masivo— y una aplicación servidora responsable de gestionar el servicio Web. Dicha aplicación también debe entender HTTP, ser capaz de codificar los archivos binarios en formato MIME y, lo que realmente la diferencia de la aplicación cliente,

```
Content-Type: image/pjpeg; name="menu.jpg"
Content-Transfer-Encoding: base64

/9j/4AAQSkZJRgABAQEAYABgAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsL
DBkSEw8UHRofHh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/
2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIy
MjIyMjIyMjIyMjIyMjIyMjIyMjL/wAARCAAMAAwDASIAAhEBAxEB/8QA
HwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUF
BAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkK
FhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1
dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXG
x8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEB
AQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAEC
AxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRom
JygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOE
hYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU
1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDk2t7crM/m
+QhjL28e4eHJ0cHJOTk9eccYHA0kIl1QquCRnLHJOTk9eccYHA0kIl1QquCR
bTy2d1FcwPtmicSRtgHawORwfcVFge/514yyk7d9Rkgd0nc4wsOEe/l6goOE
r+vy7H//2Q==

----8CFDA75A284D5A8033E016C87CBCE897--
```

Listado 1.1. Contenido de una imagen en formato binario JPEG recodificada en formato de texto MIME dentro de un documento HTML.

el núcleo principal está compuesto por el gestor de transacciones y por el acceso al sistema de archivos en el que se encuentran las páginas HTML y demás recursos asociados.

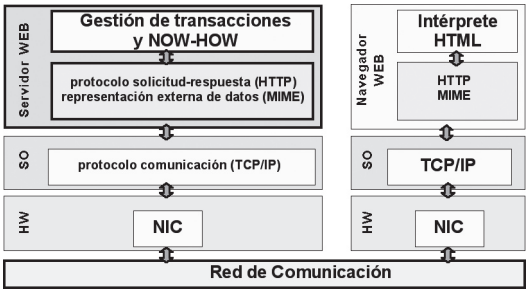


Figura 1.8. Arquitectura técnica física de una aplicación Web sobre Internet.

1.3. INTERFAZ CGI Y APLICACIONES WEB

Básicamente, una *aplicación Web* es cualquier aplicación basada en la arquitectura cliente-servidor, donde el cliente es un *navegador Web* y el servidor es un *servidor Web* (ver figura 1.3), utilizando ambos para su entendimiento el *protocolo de aplicación HTTP* (*HyperText Transfer Protocol*). Sin embargo, con las características del servidor Web que hemos estudiado hasta el momento, el servicio HTTP tan solo es capaz de proporcionar una serie de páginas estáticas, previamente definidas, que difícilmente servirían para alcanzar un objetivo tan ambicioso.

Un requisito mínimo para poder construir una aplicación a partir del servicio HTTP es contar con la capacidad de generar páginas HTML de forma dinámica en función de determinadas condiciones establecidas por el cliente. Por suerte, el servicio HTTP básico cuenta con una característica adicional que permite hacer esto: un servidor Web puede invocar una aplicación externa a petición del cliente, pasarle una serie de parámetros que éste haya indicado y, tras la ejecución de dicha aplicación, recoger su salida para, seguidamente, retransmitírsela al cliente.

Estas aplicaciones se ejecutan en el nodo servidor y pueden estar desarrolladas en cualquier lenguaje de programación, compilado o interpretado, soportado por dicho servidor. La única restricción que se les impone es que implementen un mecanismo que les permita comunicarse con el servicio Web; concretamente: acceder a los parámetros de ejecución —en caso de que éstos existan— determinados por el cliente y que su salida —en formato textual y preferiblemente en formato HTML— pueda ser recuperada por el servidor Web para pasársela al cliente que invocó la aplicación. Este mecanismo está perfectamente definido para que ambas partes la conozcan de antemano y se denomina *CGI* (*Common Gateway Interface*); razón por la cual, este tipo de aplicaciones se denominan también *aplicaciones CGI*.

En la figura 1.10 se presenta un sencillo ejemplo de solicitud HTTP `GET` en la que un cliente solicitaría al servidor la ejecución de la aplicación `busca.php` ubicada en la carpeta `/cgi-bin` y le pasa como argumento un parámetro identificado como `clave`, cuyo valor será `pdi`.

Método	URI	Cabecera
GET	/cgi-bin/busca.php?clave=pdi	

Figura 1.9. Formato de invocación de una aplicación CGI.

Otro ejemplo de solicitud de ejecución de una aplicación CGI, pero esta vez empleando el método POST, es el mostrado en la figura 1.11. En él puede observarse cómo los parámetros y sus valores, se transfieren ahora en el cuerpo de la solicitud.

Solicitud	`POST /cgi/miAplicacion.cgi HTTP/1.0`

Cabecera	`Accept: */*`
	`Connection: Keep-Alive`
	`User-Agent: Generic`
	`[línea en blanco]`

Cuerpo	`Nombre=Paco&eMail=pmacia@dtic.ua.es`

Figura 1.10. Ejemplo de solicitud HTTP utilizando el método POST.

En la figura 1.12 se muestra un sencillo escenario en el que un cliente invoca la ejecución de una aplicación CGI. Su funcionamiento, siguiendo el procedimiento que define la interfaz CGI es el siguiente:

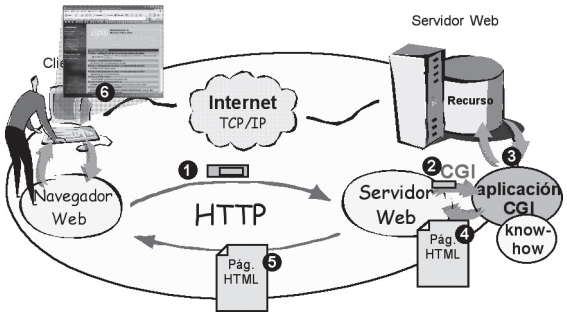


Figura 1.11. Escenario de desarrollo de un servicio HTTP mediante CGI.

❶ Cliente o navegador Web a Servidor Web. El usuario, a través de su navegador Web solicita la ejecución de una aplicación CGI al servidor. Esta solicitud puede realizarse mediante el método GET o el método POST. En el caso de una solicitud GET, los parámetros y sus valores se incluyen en la propia URI (ver figura 1.10). Con el método POST, estos mismos datos se incluyen en el cuerpo de la solicitud (ver figura 1.11).

❷ Servidor Web a Aplicación invocada. El Servidor Web copia determinados parámetros en variables de entorno (ver figura 1.13) —en Win-CGI éstas se guardan en ficheros `.INI`— y ejecuta la aplicación. Si el método de invocación era POST, además, escribe los datos del cuerpo de la solicitud (es decir, los parámetros y sus valores) en el descriptor de archivo que se utiliza como entrada estándar de la aplicación.

❸ La aplicación puede desempeñar cualquier función —como por ejemplo: realizar un cálculo o acceder a una base de datos— y tomar decisiones antes de formatear la información en HTML, para dejársela al servidor Web, el cual la enviará al cliente o navegador. Podemos considerar que el *know-how* ha pasado del servidor Web a la aplicación CGI.

❹ La aplicación, una vez realizada su labor (o a medida que la realiza), escribe en su salida estándar el resultado, procurando hacerlo en un formato válido para el cliente Web: texto en formato HTML, MIME o XML, por poner algunos ejemplos. Precisamente, el servidor Web tendrá redirigida esta salida a un descriptor propio, a través del cuál leerá este resultado.

❺ El servidor Web enviará la información al cliente Web que realizó la invocación a modo de respuesta HTTP, conformando el cuerpo de la misma a partir de los datos obtenidos en el paso anterior.

❻ Finalmente, el navegador Web, una vez recibida la respuesta, la interpretará y la mostrará (posiblemente tras recuperar mediante otras solicitudes otros recursos: gráficos, audio, etc.) al usuario de forma gráfica en su pantalla.

Variable de Entorno	Valor
REQUEST_METHOD	**GET**
QUERY_STRING	**/cgi-bin/busca.php?clave=pdi**
CONTENT_LENGTH	**29**
SCRIPT_NAME	**/cgi-bin/busca.php**
SERVER_PORT	**80**

Figura 1.12. Contenido de algunas variables de entorno utilizadas por el protocolo CGI.

La propuesta CGI aporta una capacidad de respuesta dinámica al servidor Web, de modo que las respuestas ya no van a ser meras páginas HTML estáticas, sino que pueden estar en «*función de*». Proporciona, además, una razonable libertad de elección del lenguaje de desarrollo de la aplicación CGI, que puede realizarse mediante programación en lenguajes nativos de

la plataforma y a los que probablemente estarán acostumbrados los programadores de la compañía, con lo que la curva de aprendizaje es mínima.

Como desventajas, hay que mencionar que con esta tecnología no hay relación entre el programa CGI y el servidor Web, pues ambos se ejecutan en procesos separados. Por tanto, no podemos controlar al programa CGI, es decir, no sabemos si terminó con éxito, si «*se colgó*» o si devolvió un resultado inesperado. Asimismo, como se instancia un programa CGI con cada petición que se realiza al servidor web, incluso aunque sea el mismo cliente el que la efectúe, el rendimiento no es el más óptimo y, además, se sobrecargan los recursos del nodo servidor. Finalmente, con CGI existe poca o nula portabilidad entre plataformas distintas, al escribirse el programa en un lenguaje nativo. Si el CGI se realiza en lenguaje interpretado, por ejemplo tipo Perl o PhP, aumenta la portabilidad —sólo habría que tener el intérprete correspondiente en la otra máquina— a costa de reducir el rendimiento con respecto a un lenguaje compilado.

Analicemos algunos sencillos ejemplos que nos ayuden a comprender mejor el funcionamiento de los CGIs.

En primer lugar, el listado que se muestra en la figura 1.14 corresponde a una aplicación mínima creada mediante un guión de comandos (*script*) que podrá ejecutar casi cualquier intérprete de comandos (*shell*) de Unix. En este ejemplo, el proceso escribirá en su salida estándar el código correspondiente a una página HTML que es equivalente al ejemplo de *saludo* mostrado con anterioridad en la figura 1.7. Si un cliente invoca este script mediante una solicitud HTTP del tipo: `GET /cgi/saludos.sh HTTP/1.1`, el servidor Web se encargará de que dicho guión se ejecute y recogería el resultado de su salida estándar para devolvérselo al usuario. El resultado final para este usuario coincidirá con el del ejemplo 1.7, sólo que en esta ocasión, la página no reside físicamente en un sistema de archivos, en realidad se ha generado «*al vuelo*» (o de forma dinámica).

Otro ejemplo equivalente al anterior es el presentado en el listado de la figura 1.15. En este caso, en lugar de un *script*, se ha optado por una versión escrita en *lenguaje C* que se deberá compilar para generar el correspondiente binario ejecutable. El resultado de invocar este binario por parte de un cliente será, nuevamente, análogo al de ejecutar el anterior *script* (ejemplo 1.14) o invocar la página HTML del ejemplo 1.7.

Un ejemplo un poco más «*interactivo*» es el que se presenta en la figura 1.16. En este caso, vamos a determinar qué método de invocación se ha empleado leyendo la variable de entorno `REQUEST_METHOD`. En caso de haber empleado el método `GET`, los argumentos se encontrarán en la variable

```
#!/usr/local/sh
# Comenzamos indicando quién debe interpretar el script.

# Sencillo script de shell que genera por la salida
# estándar una página HTML de saludo.

# Generamos la cabecera de la respuesta HTTP
print "Content-type: text/html"
print

# Generamos el cuerpo de la respuesta HTTP
print "<HTML>"
print "<HEAD>"
print "<TITLE>SALUDOS</TITLE>"
print "</HEAD>"
print "<BODY>"
print "<H1>¡Hola a Todos!</H1>"
print "</BODY>"
print "</HTML>"
```

Figura 1.13. Guión de comandos (*script de shell*) denominado `saludos.sh`, compatible con la mayoría de *shells* UNIX (ksh, bsh, bash), pensado para actuar como un programa CGI.

```
/* Programa C utilizado como programa CGI  *
 * generando como salida una página html   *
 * con el mensaje "Hola a todos"           */

#include <stdio.h>

main(int argc, char *argv[]) {

    printf("Content-type: text/html\n\n");
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<TITLE>SALUDOS</TITLE>\n");
    printf("</HEAD>\n");
    printf("<BODY>\n");
    printf("<H1>¡Hola a Todos!</H1>\n");
    printf("</BODY>\n");
    printf("</HTML>\n");

}  /* de main */
```

Listado 1.2. Contenido del archivo `saludos.c` con la versión en lenguaje C de la aplicación CGI "*Hola a todos*".

de entorno `QUERY_STRING`. Si, por el contrario, se ha empleado el método `POST`, el servidor Web nos habrá dejado estos argumentos en la entrada estándar. En cualquier caso, el tamaño de la cadena con los argumentos lo tendremos reflejado en la variable de entorno `CONTENT_LENGTH`.

Como resultado, este programa generará de forma dinámica el código HTML de una página en la que se mostrará cuales fueron los argumentos de la invocación.

```
/* Aplicación C escrita para ser ejecutada como        *
 * programa CGI, pensada para obtener los argumentos   *
 * pasados por el navegador Web, ya sea a través de    *
 * una invocación mediante el método GET, como el POST */

int main (void) {
    char metodo[100];
    char argumentos[1000];
    int tam;

    /* Obtenemos el método */
    strcpy(metodo, getenv("REQUEST_METHOD"));
    if (strcmp(metodo, "GET") = 0) {
       /* Argumentos en variable de entorno QUERY_STRING */
        strcpy(argumentos,getenv("QUERY_STRING"));
    }
    else {
        if (strcmp(metodo, "POST") = 0) {
            /* Argumentos en la entrada estándar */
            tam = atoi(getenv("CONTENT_LENGTH"));
            fgets(argumentos, tam + 1, stdin);
        }
        else
            printf("Error. Metodo no soportado");
    }

    /* Cabecera */
    printf("Content-type: text/html\n");
    printf("\n");

    /* Cuerpo */
    printf("<HTML>\n");
    printf("<HEAD>\n");
    printf("<TITLE>Argumentos de entrada</TITLE>\n");
    printf("<BODY>\n");
    printf("Argumentos del CGI: %s\n", argumentos);
    printf("</BODY>\n");
    printf("</HTML>\n");
}
```

Listado 1.3. Ejemplo en lenguaje C del tratamiento de las peticiones HTTP GET y POST desde una aplicación CGI.

1.3.1. Formularios Web

Tal y como ya se ha explicado, un cliente puede invocar una aplicación CGI mediante una solicitud HTML a través del método GET o POST. Sin embargo, si cada vez que un usuario desea invocar una función, debe construir a mano una de estas solicitudes, todo el sistema iría en contra de su filosofía global de transparencia y facilidad de uso de cara al usuario. Además, de forma directa, un usuario sólo podría generar solicitudes del tipo GET en la que los parámetros y valores se pasan directamente en la línea de solicitud.

Una primera alternativa consistiría en generar URIs de forma manual e incluirlas mediante etiquetas HREF. El problema es que mediante este mecanismo no se pueden modificar los valores de los parámetros de las llamadas y, por lo tanto, se convierte en una solución parcial, adecuada sólo para casos muy concretos.

Por suerte, HTML prevé esta necesidad y proporciona una serie de etiquetas que permiten definir diferentes campos para la introducción de datos dentro de nuestra página, recoger la información introducida por el usuario en los mismos y generar automáticamente la solicitud HTTP adecuada para invocar una aplicación CGI con los valores introducidos por el usuario. Este tipo de páginas HTML que incluyen esta posibilidad se denominan *formularios HTML* y las etiquetas que lo hacen posible son: `<FORM>`, `</FORM>` e `<INPUT>`.

La etiqueta `FORM` es la que determina el alcance del formulario, permite definir el método de invocación a utilizar y la aplicación CGI a la que se hará referencia.

Las etiquetas `INPUT` definen los diferentes campos y botones que presentará el formulario para la introducción de información. Se pueden definir campos de tipo texto, desplegables o de selección. También se pueden definir botones genéricos y, sobre todo, se puede definir un tipo de botón especial denominado `submit`. Este botón es uno de los más importantes del formulario pues, cuando el usuario pulsa sobre él, desencadena que el navegador Web dé por concluida la introducción de datos, los recoja y genere con ellos la solicitud GET o POST adecuada para el servidor Web.

Según esto, podríamos definir un *formulario Web* como una página Web que incluye una serie de botones y campos para la recogida de datos junto con un botón para realizar la invocación.

La figura 1.20 muestra el código HTML de una página Web que incluye un sencillo formulario de búsqueda con un campo de introducción de

texto y un botón para comenzar con la misma. En la figura 1.21 se puede observar cómo presentaría un navegador Web convencional este formulario. En este ejemplo, al pulsar sobre el botón «`buscar`» que es del tipo especial «`submit`», el navegador generaría una solicitud similar a:

```
GET /cgi-bin/busca.cgi?Nombre=Paco&Maciá
```

```
<HTML>
<HEAD>
<TITLE>Búsqueda...</TITLE>
</HEAD>

<BODY>
<H1>Formulario de búsqueda</H1>
<HR>

<FORM method="GET" action="buscar.cgi">
  Introduzca un nombre:<br>
  <INPUT NAME="Nombre"><P>
  <INPUT type="submit" value="buscar"><P>
</FORM>

<HR>
</BODY>
</HTML>
```

Listado 1.4. Formulario HTML con invocación a CGI.

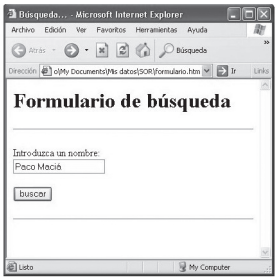


Figura 1.14. Representación gráfica por un navegador de la página HTML de la figura anterior y que incluye un formulario.

1.4. CASO DE USO: APACHE

El Servidor Web Apache es un servidor HTTP de código abierto el cual ha sido desarrollado por el grupo *Apache Software Foundation* dentro del proyecto HTTP Server (*httpd*). Este servidor ha sido creado sobre los principales sistemas operativos (Unix y Windows) haciendo frente a los servidores Web propietario de mayor uso en el mercado. Para ello sus creadores han desarrollado el servidor con los requerimientos de: seguridad, eficiencia, extensibilidad y estandarización. Estos objetivos junto con sus características de producto de código abierto han conseguido situarlo como uno de los servidores Web líderes del mercado.

1.4.1. Arquitectura modular

El servidor Web Apache se basa en una arquitectura modular que permite extender las funcionalidades y características. El servidor Apache se compone de un núcleo central con las funciones básicas y una serie de módulos que permiten añadir un conjunto de funcionalidades.

Podemos encontrar tres tipos diferentes de módulos:

- Módulos básicos. Se trata de módulos que implementan las funcionalidades básicas del Servidor Apache.
- Módulos multiproceso. Encargados de la gestión multiproceso de las peticiones.
- Módulos externos. Módulos de terceros que añaden funcionalidad adicional al servidor.

El servidor puede ser compilado para usar carga dinámica de módulos que permita compilar módulos por separado y que permita añadir módulos en cualquier momento mediante la directiva `LoadModule`. La mayoría de los módulos básicos y multiproceso no soportan la carga dinámica puesto que deben seleccionarse en tiempo de compilación.

```
# Carga del módulo correspondiente
LoadModule nombre_módulo ruta_módulo
```

Listado 1.5. Sintaxis de la directiva `LoadModule`.

Debido al carácter distribuido y flexible de los archivos de configuración, en Apache 2, la carga y la configuración de módulos puede encontrase distribuida en varios archivos. Apache permite incluir en el archivo principal de configuración (éste dependerá de la distribución utilizada y del administrador) referencias a otros archivos que también contienen datos de configuración mediante la directiva `include`. Esto permite una gestión más modular y fácil de mantener. En la distribución que se incluye por defecto de Apache 2 para la distribución *Debian*, la carga de módulos se puede realiza utilizando archivos con extensiones `.load` y `.conf`. El archivo `.load` se encarga de la carga del módulo correspondiente y el archivo `.conf` de la configuración necesaria para que el módulo funcione correctamente. Los archivos de carga y configuración de los módulos se suelen ubicar en `/etc/apache2/mods-enabled`. La directiva `include` en el archivo de configuración `apache2.conf` permite incluir estos archivos para cargar los módulos correspondientes.

```
… …
Include /etc/apache2/mods-enabled/*.load
Include /etc/apache2/mods-enabled/*.conf
```

Listado 1.6.
Ejemplo de uso de la directiva `Include` en el archivo `apache2.conf` .

En el caso práctico de *Apache Tomcat* se mostrará cómo cargar el módulo que permite conectar el servidor Web *Apache* y *Tomcat*.

```
# Carga del modulo correspondiente
LoadModule jk_module /usr/apache2/modules/mod_jk.so
```

Listado 1.7. Ejemplo de archivo `*.load` para cargar el módulo `mod_jk`.

El archivo `*.conf` será dependiente del módulo que se quiera cargar y se debe consultar el manual del módulo para su configuración.

1.4.1.1. Módulos multiprocesos

Apache provee la opción de configurar la distribución de los procesos que atenderán a las peticiones realizadas por los clientes. Esta característica permite optimizar el servidor Web en función del uso que se le vaya a dar o de la plataforma sobre la cual se instale. Algunos de los tipos de configuración de la distribución de los procesos son:

- *Prefork*. El proceso principal crea un conjunto de hijos que atienden las peticiones. Cada proceso hijo atiende una única petición de manera simultanea. El número de procesos hijos se establece entre un mínimo y un máximo.

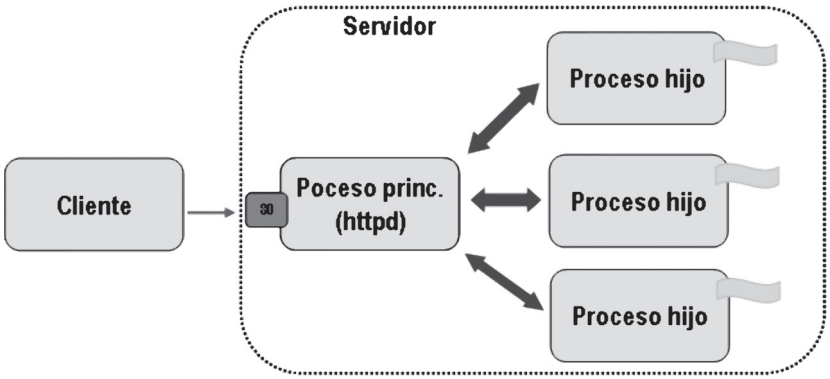


Figura 1.15. Modelo prefork.

- *Worker*. A diferencia del anterior, los procesos hijos pueden atender varias peticiones simultáneamente, es decir, se trata de un modelo híbrido de proceso e hilos. Cada proceso hijo puede establecer más de un hilo. Cada hilo dentro de un proceso atenderá a una petición. Esta gestión basada en multi-hilos es más eficiente que si se basara en procesos puesto que los hilos son más ligeros, en términos de procesamiento, que los procesos. Esta técnica establece el número máximo y mínimo de hilos que optimicen el rendimiento del servidor y se crean y destruyen los procesos hijos para mantener esta medida.

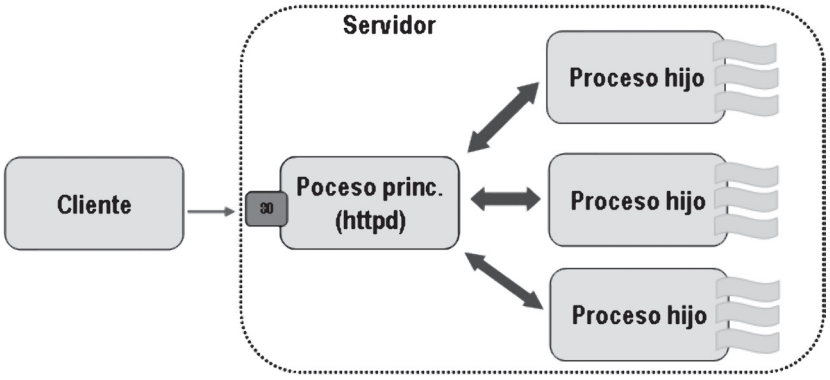


Figura 1.16. Modelo worker.

- *Perchild*. En esta técnica el número de procesos hijos que se establece es fijo. Una de las características que diferencia a esta técnica es que permite relacionar cada proceso hijo con un identificador de usuario o un grupo y atender las peticiones relacionadas con éstos. Esto permite asignar usuarios y grupos a diferentes servidores virtuales. Además, permite establecer la creación de hilos en función de la carga esperada en cada servidor virtual.

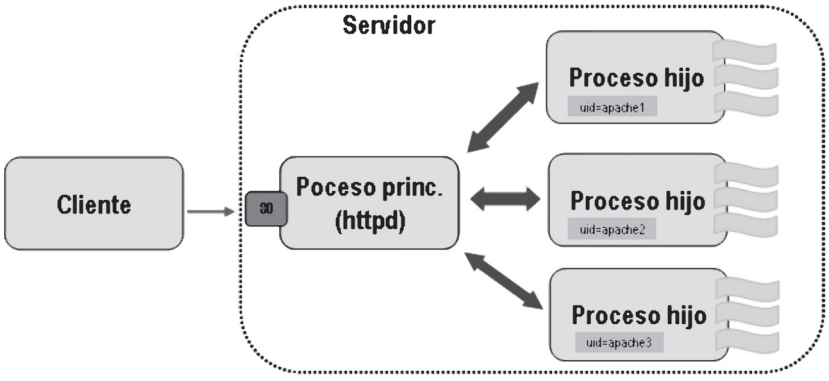


Figura 1.17. Modelo perchild.

- *Winnt*. Esta técnica está optimizada para entornos Windows. Existe un proceso padre y un proceso hijo que establece varios hilos de ejecución. Para optimizar utiliza funcionalidades del propio sistema Windows.

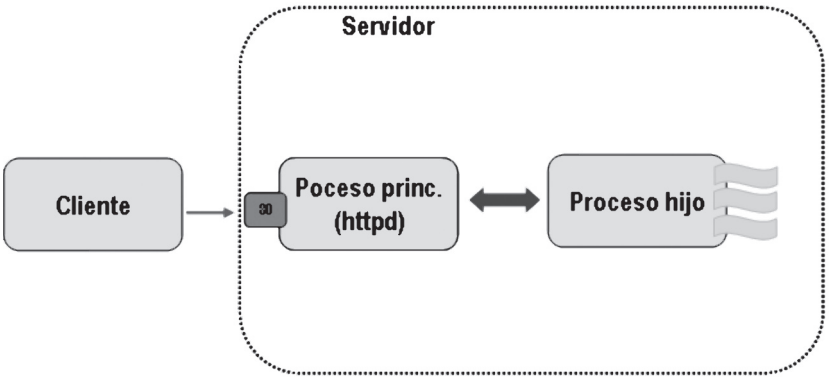


Figura 1.18. Modelo winnt.

La selección del módulo de multiproceso deseado se realiza en la opción `configure` cuando se compila el servidor mediante el parámetro `--with-mpm=nombre_del_módulo`. Si no se indica esta opción, por defecto se elegirá el módulo más adecuado para el sistema operativo sobre el cual se está trabajando. Únicamente se puede seleccionar un módulo de multi-procesamiento en un determinado momento y por lo tanto solo habrá un módulo mpm cargado. Para cada caso existen una serie de directivas que permiten configurar las características de los procesos e hilos (consultar la documentación `http://httpd.apache.org/docs/2.2/mod/directives.html`).

1.4.2. Instalación de Apache en Linux

La instalación de Apache 2 se realizará sobre la distribución de Linux *knoppix 5.1* está basada en Debian.

1.4.2.1. Instalación desde el código fuente

En primer lugar se debe descargar la versión de Apache 2.2.4 desde la página oficial del grupo Apache `http://httpd.apache.org/download.cgi`.

De esta forma, para instalar el servidor Apache en Linux se debe compilar los archivos fuentes. Descargamos los archivos de código fuente `httpd-2.2.4.tar.gz`.

Una vez descargados los archivos fuentes se procederá a su descompresión. Para extraer el código fuente del archivo *.tgz* descargado, se debe ejecutar los siguientes comandos:

```
$ gzip -d httpd-2_2_4.tar.gz
$ tar xvf httpd-2_2_4.tar
```

Listado 1.8. Comandos para la extracción de los archivos fuentes del servidor Apache.

La ejecución de estos comandos creará un directorio en la misma ubicación donde se encuentra el archivo que se ha descomprimido, incluyendo el código fuente de la distribución deseada (`httpd-2.2.4`).

Una vez extraído el código fuente se debe cambiar la ubicación al directorio `httpd-2.2.4` donde se encuentran dicho código.

Es posible cambiar parámetros de configuración para la instalación del Servidor Apache (directorio de instalación, módulos a cargar, etc.) En este caso se configura con los parámetros por defecto. Para ello se debe utilizar el *script* llamado `configure` que se encuentra situado en el directorio raíz de la distribución que se ha extraído, `httpd-2.2.4`. Este *script* soporta un gran número de opciones de configuración que se puede consultar en la documentación oficial de apache (`http://httpd.apache.org/docs`).

```
$ cd httpd-2.2.4
$ ./configure
```

Listado 1.9. Configuración del servidor Apache 2 previo a su compilación.

Una vez configurado los parámetros se procede a la compilación del Servidor.

```
$ make
```

Listado 1.10. Comandos para la compilación del servidor Apache 2.

Una vez se ha compilado el servidor (puede llevar varios minutos) se procede a su instalación mediante el siguiente comando. Para realizar esta operación se debe contar con privilegios de súper-usuario (*root*). Es posible que al cambiar de usuario os traslade a la ubicación del nuevo usuario. Se deberá volver al directorio `httpd-2.2.4`.

```
$ su -
$ ...
$ make install
```

Listado 1.11. Comandos para la instalación del servidor Apache.

Una vez ejecutado el comando mostrado en Listado 1.27 el servidor Apache, generalmente, se habrá instalado en el directorio por defecto (en este caso en `/usr/local/apache2`). Una vez instalado se puede comprobar su correcto funcionamiento utilizando el *script* incluido en la distribución, como se indica en el listado 1.28, y que facilita la gestión del demonio generado `httpd`.

```
$ /usr/local/apache2/bin/apachectl start
$ netstat -ln
$ /usr/local/apache2/bin/apachectl stop
```

Listado 1.12. Comandos para lanzar y parar el servidor Apache.

La primera secuencia de comandos del Listado 1.28 permite arrancar el Servidor Apache. Para comprobar que ha arrancado correctamente se utiliza la segunda línea de comandos. Esta orden muestra los servicios que se encuentran escuchando a la espera de peticiones (en el caso de Apache se situará en el *puerto 80*). Una prueba más sencilla consiste en abrir un navegador y acceder a la dirección de la máquina donde se ha instalado (`http://ip_servidor`). De esta manera se accederá a la página por defecto del servidor Apache.

Una vez comprobado el correcto funcionamiento se puede parar el servidor para proceder a su configuración como se indica en la tercera línea del listado 1.28.

1.4.2.2. Instalación desde un paquete binario

Una manera quizás más sencilla de instalar el servidor Web Apache consiste en descargar una distribución binaria preparada para la distribución de Linux con la que se vaya a trabajar, en nuestro caso la distribución Knoppix, la cual se basa en Debian. En este caso la distribución Knoppix elegida (distribución *5.1*) incluye por defecto la instalación del servidor Apache 2, pero si no sucediese así, al estar basada en Debian, se debería buscar para instalar el paquete preparado para dicha distribución (extensión *.deb*).

Para instalar Apache Web a partir de un paquete *.deb* se puede seguir dos procedimientos (en ambos casos se debe ser el usuario *root*). En el primero debemos obtener el paquete *.deb* que contiene el servidor Web Apache, por ejemplo de `www.debian.com`. Una vez descargado el paquete se debe utilizar la herramienta `dpkg` como se indica en el listado 1.29.

```
$ dpkg –i ruta_donde_está_el archivo_deb/apache2-2.2.3-4_all.deb
```

Listado 1.13. Comandos para la instalación del servidor Apache desde paquete Debian.

El inconveniente de este procedimiento de instalación radica en que se puede encontrar problemas de dependencias con otros paquetes y el proceso se puede volver un poco más complejo.

El segundo proceso de instalación desde un paquete *.deb* consiste en el uso de la herramienta `apt-get`. Esta herramienta permite la instalación de paquetes, en general para cualquier distribución de Linux, instalando las dependencias que solicite el proceso. La herramienta utiliza un archivo de selección de los *mirrors* de los cuales obtendrá los paquetes. Este archivo puede ser modificado y se llama `sources.list`. Para la distribución que se está usando, este archivo se encuentra ubicado en el directorio `/etc/apt`. En el listado 1.30 se puede ver el uso de `apt-get` para instalar el paquete de Apache 2.

```
$ apt-get install apache2
```

Listado 1.14. Comandos para la instalación del servidor Apache con apt-get.

Como se ha comentado anteriormente en la distribución de Knoppix que se está utilizando el servidor Web Apache ya viene instalado por defecto, por lo tanto trabajaremos con esta distribución.

1.4.3. Configuración básica de Apache

La configuración del servidor Apache 2 se realiza sobre el archivo `httpd.conf` según la documentación oficial. No obstante algunas distribuciones de Linux, como la usada en los ejemplos utiliza el archivo `apache2.conf`, en este caso ubicado en `/etc/apache2`. Sin embargo, si se revisa el archivo `apache2.conf` se puede observar que a través de la directiva `include` contiene una referencia al archivo `httpd.conf`. Como sucede en esta distribución, el archivo de configuración puede cambiarse de nombre e indicar al arrancar el servicio de Apache, con la opción `-f`, el nombre y la ruta del archivo de configuración, sin necesidad de que sea el archivo de configuración por defecto de la distribución.

```
$ httpd -f archivo_configuración
```

Listado 1.15. Selección de archivo de configuración lanzando Apache.

Apache se configura incluyendo en el archivo principal de configuración, directivas en texto plano, generalmente del tipo directiva valor. Como se comentó anteriormente, Apache 2 permite, por medio de la directiva `Include` separar en archivos más pequeños y manejables el contenido del archivo de configuración principal.

A continuación se describen las principales directivas de Apache 2.

La primera consideración en la configuración del un servidor Web es el establecimiento del *puerto* o *puertos* que vamos a disponer para la escucha de las peticiones realizadas por el cliente. En Apache 2, la directiva que se utiliza es `Listen`.

La directiva `Listen` indica las direcciones IP y los *puertos* en los que debe escuchar Apache; por defecto, el servidor responde a las peticiones que se reciban en cualquier dirección IP de las interfaces de red. Su uso es ahora obligatorio. Si solamente se especifica un número de puerto, el servidor escuchará en ese *puerto*, en todas las interfaces de red que posee. Si se especifica una dirección IP y un *puerto*, el servidor escuchará solamente en esa dirección IP y en ese puerto.

Se pueden usar varias directivas `Listen` para especificar varias direcciones y *puertos de escucha*. El servidor responderá a peticiones de cualquiera de esas direcciones y *puertos*.

```
Listen 80
Listen 8080
```

Listado 1.16. Configuración del *puerto* de escucha de Apache.

En el listado 1.32 se puede ver la configuración para que el servidor Apache acepte peticiones a través de los *puertos 80* (*puerto* destinado por defecto a los servidores Web o HTTP) y *8080*.

Se podría limitar además para que el servidor aceptase peticiones en el conjunto de *puertos* determinado y dirigidas a la direcciones IP indicadas en la directiva `Listen`. En el listado 1.33 las peticiones que se envíen a las direcciones `192.168.1.10` a través del *puerto 80* o las que se envíen a la dirección `192.168.2.10` a través del *puerto 8080* serán aceptadas.

```
Listen 192.168.1.10:80
Listen 192.168.2.10:8080
```

Listado 1.17. Configuración de puertos y direcciones de escucha.

En la distribución que se está utilizando, la configuración de los *puertos* se encuentra en `/etc/apache2/ports.conf` y en el archivo `apache2.conf` se incluye mediante la directiva `Include`.

En la configuración de Apache, se debe indicar el usuario y grupo con el que se ejecutarán los procesos hijos del servidor. Para ello se dispone de las directivas `user` y `group`. Por ejemplo, para que un servidor Web lance sus procesos con el usuario de Linux *nobody* y el grupo *nobody*, se deberá insertar las directivas:

```
User nobody
Group nobody
```

Listado 1.18. Configuración del grupo y usuario para la ejecución de Apache.

Como se ha comentado, un servidor Web se dedica a servir páginas HTML que se encuentran *bajo* un determinado directorio. Para indicar a un servidor cuál es el directorio donde se encuentran los documentos que debe mostrar, se utiliza la directiva `DocumentRoot`. Si se desea que un servidor *sirva* páginas almacenadas en el directorio `/var/www`, se indicará de la siguiente manera:

```
DocumentRoot /var/www
```

Listado 1.19. Configuración del directorio principal del servidor.

No obstante, esta directiva en la distribución que se está utilizando para mostrar los ejemplos no se encuentra declarada en el archivo principal, `apache2.conf`. En su lugar, se encuentra definida en el archivo de configuración, `000-default`, en el directorio `/etc/apache2/sites-enabled/`, y es el archivo `apache2.conf` quien se encarga de incluir estos archivos que activan los distintos *sitios* (listado 1.36).

```
Include /etc/apache2/sites-enabled
```

Listado 1.20. Ejemplo de la configuración distribuida de Apache.

En el siguiente listado se puede ver el contenido del archivo `000-default`. En este ejemplo se utiliza la directiva `Directory` la cual permite agrupar un conjunto de directivas para un directorio o directorios determinado del sistema de archivo local. Las directivas agrupadas en el marco de un directorio se extienden a sus subdirectorios y prevalecen sobre las directivas globales declaradas para el servidor Apache. En el listado mostrado abajo, se puede observar que como se aplican unas directivas genéricas al directorio raíz de Linux. Estas directivas serán heredadas por todos los directorios del sistema de archivos, a no ser que, explícitamente para un directorio se sobrescriban. Lo mismo sucede con aquellas directivas de tipo global que se encuentren fuera de la cláusula `<Directory>`. Esta característica permite configurar propiedades específicas para cada directorio que almacene recursos Web del servidor.

```
NameVirtualHost *
<VirtualHost *>
     DocumentRoot /var/www
     <Directory />
          Options FollowSymLinks
          AllowOverride None
     </Directory>
     <Directory /var/www/>
          Options Indexes FollowSymLinks Multiviews
          AllowOverride None
          Order Allow,Deny
          Allow from all
     </Directory>
… …
</VirtualHost>
```

Listado 1.21. Ejemplo de uso de la directiva `Directory`.

Junto a la cláusula `Directory` o `Location` (similar a `Directory` pero hace referencia a una *URL* en lugar de al sistema de archivos) se puede utilizar la directiva `Alias`. Esta directiva permite el mapeo de *URL* con direcciones del sistema de archivo local donde se almacenen recursos Web. Se puede indicar una dirección local que no este incluida en la dirección especificada por el `DocumentRoot`. El formato de esta directiva se describe a continuación:

```
Alias URL_path file_path/directory_path
```

Las peticiones cuyas *URLs* desde la dirección base (`http://ip:puerto`) coincidan con la *URL_path* definida por `Alias`, en este caso `/ejemplo`, accederán al recurso ubicado en el directorio indicado como segundo parámetro de la directiva (`/home/knoppix`).

```
DocumentRoot /var/www
……
Alias /ejemplo "/home/knoppix/"
<Directory /home/knoppix/>
Options Indexes FollowSymLinks Multiviews
    AllowOverride None
    Order Allow,Deny
    Allow from all
</Directory>
```

Listado 1.22. Ejemplo de uso de la directiva `Alias`.

En la figura 1.26 se presenta una petición de acceso al recurso `index.html` ubicado en el directorio `/home/knoppix` mediante la *URL* `http://localhost/ejemplo`.

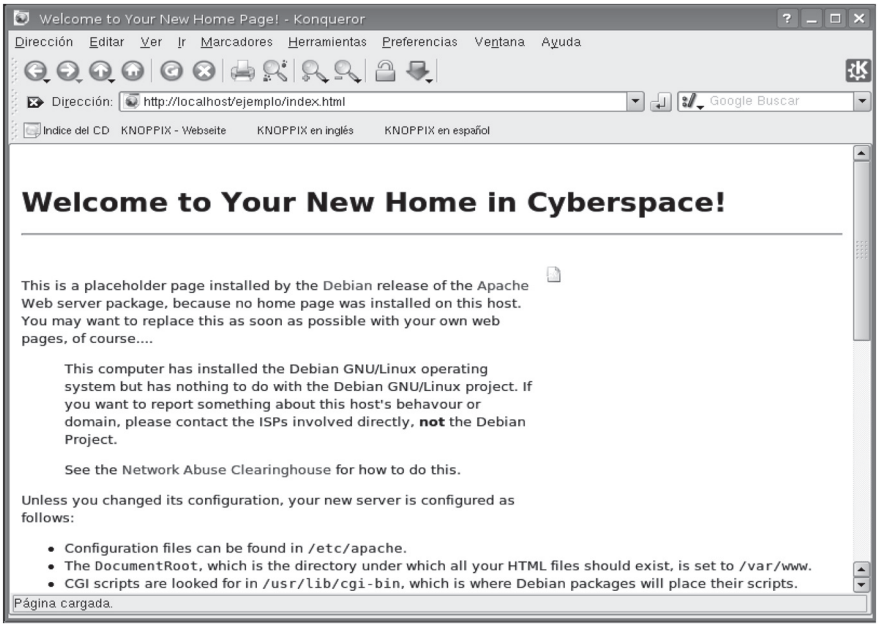


Figura 1.19. Acceso a recursos mediante `Alias`

La directiva `Alias` facilita el acceso a complejas direcciones y oculta la ubicación real donde se encuentra un determinado recurso.

Así mismo, al servidor Web también se le debe indicar a partir de qué directorio se encuentran los archivos necesarios para el funcionamiento del servidor como por ejemplo, `srm.conf, httpd.conf`, los archivos de logs, ... Para ello dispone de la directiva `ServerRoot`, con la que se le indica esta información. Cualquier directiva en la que se indique una ruta no absoluta será relativa a la indicada en la cláusula `ServerRoot`.

```
ServerRoot /etc/apache2
```

Listado 1.23. Ejemplo de uso de la directiva `ServerRoot`.

Para indicar exactamente dónde se encuentran los archivos de logs, se puede usar las directivas `ErrorLog` y `CustomLog` con las que, respectivamente, se indican los archivos de error y acceso. Si indicamos un *path absoluto*, no se tiene en cuenta la directiva `ServerRoot` para obtener la localización exacta de los archivos. Por ejemplo, para que el archivo donde se guardan los errores que ocurren en el servidor sea `/var/log/httpd/errores.log`, la directiva a usar es:

```
ErrorLog /var/log/httpd/errores.log
```

Listado 1.24. Ejemplo de configuración de la directiva `ErrorLog`.

Si queremos que el registro de accesos se guarden en el archivo `/var/log/httpd/accesos.log`, la directiva usada será:

```
CustomLog /var/log/httpd/accesos.log combined
```

Listado 1.25. Ejemplo de configuración de la directiva `CustomLog`.

El segundo argumento de la directiva indica el formato de las líneas de *log* dentro del archivo indicado.

Cuando se introduce una *URL* del estilo `http://nombre_servidor:puerto/directorio_local`, en el que no se especifica la página HTML a mostrar, el servidor puede actuar de 2 maneras diferentes. Por un lado, puede mostrar un listado del directorio (no recomendable) o, por otro lado, puede mostrar el contenido de una página HTML por defecto. Para conseguir esta última opción, se debe definir la directiva `DirectoryIndex`, de tal forma que cuando no se indique una página en concreto, se muestre una por defecto (normalmente `index.html`) que debe residir en el directorio al que se accede (en este caso el directorio será *directorio_local* y estará incluido en la ruta indicada en la directiva `DocumentRoot`). A continuación se muestra un ejemplo del uso de esta directiva:

```
DirectoryIndex index.html index.htm index.php index.phtml index.js
```

Listado 1.26. Ejemplo de configuración de la directiva `DirectoryIndex`.

En la tabla inferior se describe de forma muy breve otras directivas de configuración básicas de Apache 2.

DIRECTIVA	DESCRIPCIÓN
ServerAdmin (dirección-email | URL)	Indica la dirección de correo electrónico del administrador del servidor
ServerName ([esquema://]dominio[:port])	Indica el nombre y el puerto para identificar al servidor
Options ([+|-]opción[[+|-]opción] …)	Indica las características disponibles en un directorio
TypesConfig (ruta_archivo)	Indica la ubicación del archivo de configuración de los tipos MIME
DefaultType (Tipo_MIME)	Tipo Mime por defecto que el servidor enviará si no encuentra el adecuado
AddDefaultCharset (On|Off|Juego_caracteres)	Juego de caracteres por defecto para devolver en una respuesta del servidor cuando el Content-Type es text/plain o text/html
Timeout (Tiempo_en_segundos)	Tiempo máximo de espera entre una petición y su respuesta
KeepAlive (Off | On)	Permite peticiones persistentes y sesiones HTTP duraderas en tiempo. Permite mútiples peticiones sobre la misma conexión TCP.
MaxKeepAliveRequest (Número_peticiones)	Número máximo de solicitudes por conexión persistente
KeepAliveTimeout (Tiempo_en_segundos)	Tiempo máximo de espera en una conexión persistente

Tabla 1.1. Directivas de configuración del servidor Apache 2.

1.4.4. Autenticación, autorización y control de acceso

Es necesario gestionar el acceso de usuarios y máquinas al servidor Web y controlar los accesos a determinados directorios y recursos. Apache ofrece una serie de directivas que permiten controlar quién quiere acceder al servidor y dónde quiere acceder.

Las directivas que ofrece Apache 2 para controlar el acceso deben ir incluidas en una cláusula `<Directory>` o en un archivo `.htaccess` en el directorio en el que se quiera configurar el acceso. Este es el nombre por

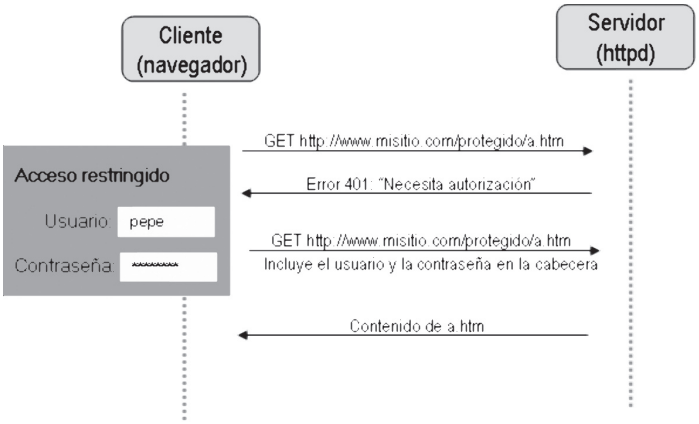


Figura 1.20. Solicitud de autenticación.

defecto pero se puede modificar cambiando la directiva `AccessFileName`. Las directivas indicadas en el archivo afectan tanto al directorio donde se encuentre ubicado como a los subdirectorios. Para obtener más información sobre este archivo se puede consultar la documentación oficial de Apache en `http://httpd.apache.org/docs/2.2/howto/htaccess.html`.

De aquí en adelante, se configurará el control de acceso utilizando la directiva `<Directory/>`.

Lo primero que se debe hacer es proteger el acceso a un directorio mediante contraseñas. Se debe crear un archivo para almacenar los usuarios y contraseñas y que no sea accesible desde la Web. Para crear dicho archivo de usuarios y contraseñas, se dispone del comando `htpasswd`. Con dicho comando, se puede crear un archivo que contenga los diferentes usuarios Apache del sistema. Por ejemplo, la ejecución del siguiente comando, crea el archivo `usuarios` con el usuario `alu1`. La ejecución de este comando solicitará por la terminal que se introduzca la contraseña del usuario.

La ayuda de este comando indica los parámetros que se pueden utilizar y su función (`man htpasswd`). La opción `-c` se indica cuando se va a crear por primera vez dicho archivo.

Para proteger, por ejemplo, el directorio `/var/www/privado` y que sólo el usuario `alu1` pueda acceder a él, el administrador del servidor Web debe introducir las siguientes directivas en el archivo de configuración. En este caso, para la distribución utilizada, se podría introducir en el archivo `/etc/apache2/sites-enabled/000-default`.

```
$htpasswd -c /etc/apache2/usuarios alu1
New password: mipassword
Re-type new password: mipassword

#Para añadir el uuario alu2, deberemos ejecutar la orden:
$htpasswd /etc/apache2/usuarios alu2
New password: mipassword2
Re-type new password: mipassword2
```

Listado 1.27. Ejemplo de creación de usuarios de Apache.

```
NameVirtualHost *
<VirtualHost *>
     DocumentRoot /var/www
     <Directory />
          Options FollowSymLinks
          AllowOverride None
     </Directory>
     <Directory /var/www>
     ……..
     </Directory>
     <Directory /var/www/privado>
              AuthName "Acceso restrinigido"
              AuthType Basic
              AuthUserFile /etc/apache2/usuarios
              Require valid-user
     </Directory>
</VirtualHost>
```

Listado 1.28. Ejemplo de configuración de la seguridad en un directorio.

La directiva `AuthName` indica el dominio en el cual el usuario se va a *autenticar*. Normalmente el navegador Web o cliente que solicita la autenticación presenta este dominio al usuario, de esta forma el usuario puede saber en que dominio se va a identificar y utilizar la contraseña adecuada si posee varias.

La directiva `AuthType` define el método que se utilizará para autenticar al usuario. En este caso es una autenticación básica sin ningún tipo de encriptación ni seguridad extra. Se debe tener en cuenta esta característica si la información que viaja a través de la red es delicada.

La directiva `AuthUserFile` indica la ubicación del archivo que contiene la información necesaria para autenticar al usuario y que fue creado mediante la utilidad `htpasswd`.

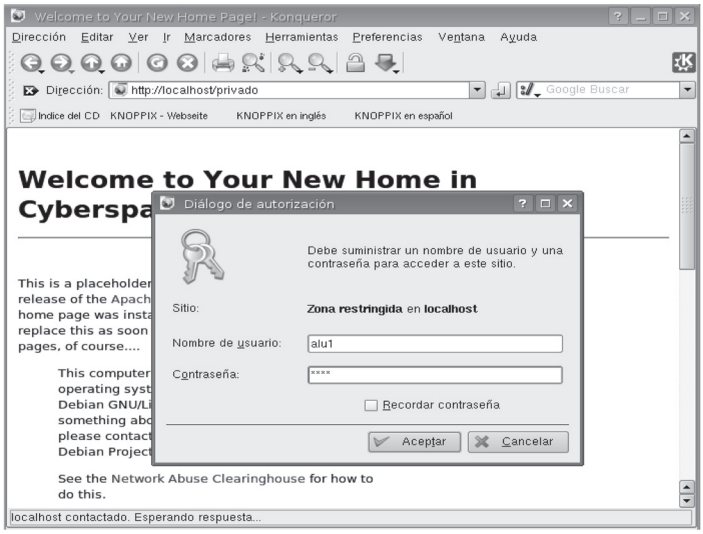


Figura 1.21. Solicitud de autenticación.

Por último la directiva `Require` establece el criterio de autorización indicando a qué usuario se le permite el acceso. En el ejemplo están autorizados todos aquellos usuarios que sean válidos a la hora de la autenticación, es decir, todos los usuarios contenidos en el archivo `usuarios`. Para autorizar a un usuario único simplemente indicaríamos su *login* el la directiva `Require`.

Existe otra forma de autorizar el acceso por grupos mediante la directiva `AuthGroupFile`. Para ello primero se debe crear un archivo de tipo texto que asocie cada nombre de grupo con una lista de usuarios.

Los usuarios del grupo deben estar incluidos en el archivo de usuarios. A continuación se muestra cómo se utilizaría la directiva `AuthGroupFile`.

```
Grupo1: alu1 alu2 alu3
Grupo2: alu4 alu5
```

Listado 1.29. Ejemplo archivo de grupos.

En este caso la directiva `AuthGroupFile` indica la ubicación del archivo que define los grupos. La directiva `Require` indica que el acceso se realizará por grupos y únicamente los usuarios del `Grupo1` están autorizados para acceder al recurso.

```
NameVirtualHost *
<VirtualHost *>
     DocumentRoot /var/www
     ……
     </Directory>
     <Directory /var/www/privado>
              AuthName "Acceso restrinigido"
              AuthType Basic
              AuthUserFile /etc/apache2/usuarios
              AuthGroupFile /etc/apache2/groups
              Require group Grupo1
     </Directory>
</VirtualHost>
```

Listado 1.30. Ejemplo de seguridad por grupos.

Otra opción de Apache, es permitir el acceso de clientes a los recursos en base al origen de su petición dependiendo de, por ejemplo, su dirección IP o el nombre de la máquina que solicita el acceso. Para ellos se pueden utilizar las directivas `Allow` y `Deny`. La directiva `Order` va unida a las anteriores e indica el orden en el cual ejecutar las anteriores. Por ejemplo, para denegar a todos los clientes que no se conecten desde la IP `127.0.0.1`, se introduciría en el archivo `/etc/apache2/sites-enabled/000-default` el fragmento del listado 1.47:

```
<Directory /var/www/privado>
      Order deny,allow
      Deny from all
      Allow from 127.0.0.1
</Directory>
```

Listado 1.31. Ejemplo de seguridad por acceso de equipo.

1.4.5. Configuración de hosts virtuales

Una de las características que ofrece Apache 2 es la posibilidad de crear *hosts virtuales*. Los *hosts virtuales* permiten configurar diferentes sitios Web en un único equipo servidor. De esta manera, para una misma máquina y un mismo servidor con tres proyectos Web, cada uno en un

directorio del `DocumentRoot` diferente, se les podría asignar tres nombres de dominio independientes.

```
http://localhost/asignaturas → http://www.asignaturas.com
http://localhost/alumnos     → http://www.alumnos.com
http://localhost/servicios   → http://www.servicios.com
```

Algunas de las ventajas que pueden ofrecer los *hosts virtuales* son: ahorro en infraestructuras teniendo en cuenta que los recursos se encuentra localizados en un único servidor, configuración compartida puesto que habrá directivas que afecten a todos los *hosts virtuales* o administración centralizada ya que comparten ubicación de archivos de configuración.

Las directivas para crear *hosts virtuales* se configuran en el archivo de configuración `httpd.conf` pero como el archivo principal, como se comentó anteriormente, depende de la distribución, en este caso se declararán en el archivo `/etc/apache2/sites-enebled/000-default`. También podría configurarse directamente en el archivo `apache2.conf` pero no sería tan legible. Existen tres formas de crear *hosts virtuales*:

- *Basadas en nombres*. En este caso existe un único servidor principal y existen diferentes nombres de dominio (DNS) que apuntan al servidor principal. Este sería el ejemplo ilustrado anteriormente.

```
NameVirtualHost 192.168.1.21:80
<VirtualHost 192.168.1.21:80>
      DocumentRoot /var/www
      ServerName www.miservidor.com
      ServerAdmin admin@miservidor.com
      ……
</VirtualHost>
<VirtualHost 192.168.1.21:80>
      DocumentRoot /var/www/ejemplos
      ServerName www.miservidor2.com
      ServerAdmin admin@miservidor2.com
      ……
</VirtualHost>
      <VirtualHost 192.168.1.21:80>
      DocumentRoot /var/www/recursos
      ServerName www.miservidor3.com
      ServerAdmin admin@miservidor3.com
      ……
</VirtualHost>
```

Listado 1.32. Ejemplo de configuración de *hosts virtuales* basados en nombre.

De esta forma para la misma IP de una máquina y el mismo servidor Web se tiene la posibilidad de acceder mediante tres nombres de dominio diferentes. Dentro de cada *host virtual* se pueden usar las directivas que creamos convenientes. La directiva `NameVirtualHost` únicamente se utiliza en este caso de *hosts virtuales*.

- *Basadas en IP*. Sucede cuando existe un único equipo con varias direcciones IP asignadas y un único servidor Web. En este caso se pueden crear un *host virtual* asignado a cada dirección IP. En el caso de que tuviéramos dos direcciones IP asignadas la configuración se podría realizar como se muestra en el ejemplo de listado 1.49.

```
<VirtualHost 192.168.1.21:80>
     DocumentRoot /var/www
     ServerName www.miservidor.com
     ServerAdmin admin@miservidor.com
     ……
</VirtualHost>
<VirtualHost 192.168.1.22:80>
     DocumentRoot /var/www/ejemplos
     ServerName www.miservidor2.com
     ServerAdmin admin@miservidor2.com
     ……
</VirtualHost>
```

Listado 1.33. Ejemplo de configuración de hosts virtuales basadas en IP.

- *Varios servidores principales*. Otra forma de soportar múltiples *hosts* consiste en la ejecución de varios demonios del servidor Web Apache (`httpd`), uno por cada nombre de dominio. Cada instancia del *demonio* que se lanza tiene asignada una dirección IP diferente. Cada una de las instancias tendrá un archivo independiente de configuración.

```
#httpd -f /etc/apache2/httpd1.conf
#httpd -f /etc/apache2/httpd2.conf
```

2. AMPLIACIONES EN EL CLIENTE

El navegador Web ha sido adoptado por la comunidad de usuarios de Internet como la aplicación cliente por excelencia para este entorno. No obstante, la Web se ha convertido en un medio complejo donde las aplicaciones requieren de más potencia, sin exceder los requisitos deseados por los usuarios finales (aplicaciones ligeras, mínimos requisitos de instalación y configuración, lógica de negocio, separación entre estilo y datos). Son numerosas las tecnologías que han permitido extender el navegador Web inicial para acercarlo a un contenedor en la parte del cliente que cumpla los requisitos exigidos en estas aplicaciones.

En un principio el navegador Web solamente ofrecía al cliente la posibilidad de visualizar información de tipo estática (HTML). Con la evolución de las *.COM* la competencia por presentar páginas Web más atractivas para el usuario y con una mayor funcionalidad han surgido diferentes tecnologías que permiten incorporar características multimedia en las páginas e interactuar con el usuario. De esta forma ha sido necesario ampliar las funcionalidades del navegador Web para que soporte todas estas características.

2.1. SCRIPTS EN EL CLIENTE

Una de las técnicas utilizadas para dotar de dinamismo las páginas Web en la parte del cliente ha sido el uso de código incrustado dentro del código HTML estático. Estos fragmentos de código son interpretados en tiempo de ejecución en el cliente por el navegador Web y se denominan *SCRIPT*.

De esta forma permite dotar a la página Web de efectos y funcionalidades. Las páginas de este tipo dependen de la plataforma y más concretamente del navegador sobre el cual se ejecuten. La ventaja que pueden ofrecer es la rápida respuesta ante acciones del usuario permitiendo la descarga del servidor. No obstante, se debe contemplar el enorme número de navegadores Web existentes y la compatibilidad con los diferentes lenguaje de tipo *Scripts*.

2.1.1. El lenguaje JavaScript

JavaScript es un lenguaje interpretado desarrollado por Netscape bajo el nombre de *LiveScript* y que tras formar la alianza SUN y Netscape pasó a tomar el nombre con el que hoy en día es conocido. El objetivo era crear un lenguaje con características similares, en lo que a sintaxis se refiere, a los lenguajes de alto nivel pero de uso sencillo, sin necesidad de utilizar compiladores. Simplemente, se introducía código JavaScript dentro de un documento HTML y el navegador se encargaba de interpretar este código.

Como se ha comentado, el código de lenguaje JavaScript se introduce dentro del código HTML. Existen cuatro formas diferentes de incrustar código Javascript.

- Entre las etiquetas `<script>` y `</script>`
- Especificando un archivo JavaScript en el código HTML (*.*js*).
- Especificando una expresión Javascript como valor de un atributo HTML.
- Especificando código Javascript en un evento de los controles HTML que los contemplen (ej. botón).

2.1.1.1. Etiqueta de marcado y archivos JavaScript

Para diferenciar el código HTML del código JavaScript se utiliza la etiqueta de marcado `<SCRIPT>` para indicar que comienza un fragmento de código JavaScript y `</SCRIPT>` para indicar el final.

A lo largo de un documento HTML se pueden utilizar varias veces dichas etiquetas, pero siempre que se introduzca una etiqueta de apertura se deberá introducir una de cierre.

En principio no existe ninguna norma que indique dónde se debe introducir las etiquetas JavaScript, aunque es importante conocer cómo el navegador interpreta el código para no obtener resultados no deseados.

```
<SCRIPT>
  window.alert("Hola Mundo JavaScript!!!")
</SCRIPT>
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    Hola Mundo HTML!!!
  </BODY>
</HTML>
```

Listado 2.1. Ejemplo sencillo de introducción de javascript en un documento HTML.

Cuando un cliente realiza una petición a un servidor Web el servidor responde con un documento que puede contener HTML y código JavaScript. El navegador Web comienza a analizar el código del documento HTML y va interpretando el código, bien sea JavaScript o HTML, y procediendo a su ejecución. En el ejemplo anterior primero se mostraría en pantalla una ventana con el mensaje `Hola Mundo JavaScript!!!` Posteriormente se procedería a mostrar el documento HTML.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    Hola Mundo HTML!!!
  </BODY>
</HTML>
<SCRIPT>
  window.alert("Hola Mundo JavaScript!!!")
</SCRIPT>
```

Listado 2.2. Ejemplo sencillo de introducción de JavaScript en un documento HTML.

En este último caso primero, se mostraría en pantalla el mensaje `Hola Mundo HTML!!!` Posteriormente se mostraría una ventana con el mensaje `Hola Mundo JavaScript!!!`

Para separar el código HTML del de JavaScript y poder gestionarlo y mantenerlo de una forma óptima podemos utilizar los archivos de extensión *`*.js`*. De esta forma tendremos archivos con el contenido JavaScript que anteriormente se encontraba entre las etiquetas `<script>` y `</script>`. Esta forma de incrustar JavaScript nos permite, además, reutilizar funcionalidades comunes a varias páginas. Es posible combinarlo con la forma de introducir JavaScript anteriormente mencionada.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    Hola Mundo HTML!!!
  </BODY>
</HTML>
<SCRIPT SRC="saludo.js">
  ...
</SCRIPT>
```

Listado 2.3. Ejemplo sencillo de introducción de JavaScript en un documento HTML mediante archivos **.js*.

```
window.alert("Hola Mundo JavaScript!!!")
```

Listado 2.4. Contenido del archivo `saludo.js`.

El atributo `SRC` puede contener una *URL* relativa o absoluta.

En estas dos formas de incrustar código JavaScript, junto con la tercera manera, el navegador en cuanto recibe el documento desde el servidor analiza el código y directamente va interpretándolo, mostrando el resultado al usuario.

2.1.1.2. Valores de atributos y eventos

Otra de las posibles maneras de incrustar código JavaScript es introduciéndola dentro de las etiquetas del código HTML, bien como valores de atributos, bien como valores de eventos de los controles.

En el primer caso el valor del atributo HTML es creado de manera dinámica de tal forma que cuando el navegador vaya a interpretar el código HTML interpretará la expresión JavaScript asociada.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <HR WIDTH=”&{barWidth};%” ALIGN=”LEFT”/>
    Hola Mundo HTML!!!
  </BODY>
</HTML>
```

Listado 2.5. Ejemplo de código HTML dinámico mediante JavaScript.

En el segundo caso se produce como respuesta a una acción realizada por el usuario, un evento. Mediante JavaScript es posible controlar las

```
<SCRIPT>
  function saludar()
  {
    Document.alert(“Hola Mundo Javascript”);
  }
</SCRIPT>
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <input type=button value=enviar
    onclick="saludar()"/>
    Hola Mundo HTML!!!
  </BODY>
</HTML>
```

Listado 2.6. Ejemplo de evento en JavaScript.

acciones que realiza un usuario al interactuar con la página Web y de esta forma realizar una funcionalidad ante el comportamiento del usuario. Para conseguir esta segunda forma, el código JavaScript debe ser introducido en una serie de atributos del código HTML.

Cuando se pulsa el control de tipo botón con la leyenda `enviar` se llama al evento `onclick` del control que realiza una llamada a la función JavaScript `saludar()`.

2.2. HTML DINÁMICO: DHTML

DHTML (acrónimo de *dynamic HTML*) no es un lenguaje de *Script* como Javascript. Se trata de la unión de varias tecnologías para crear sitios Web dinámicos e interactivos. DHTML utiliza HTML estático, un lenguaje de tipo *Script* de lado del cliente (como JavaScript), un lenguaje de definición de estilos para la presentación (por ejemplo *CSS*) y el modelo de objetos de documentos (*DOM*). DHTML no es una tecnología por sí misma si no que es la unión de un conjunto de éstas.

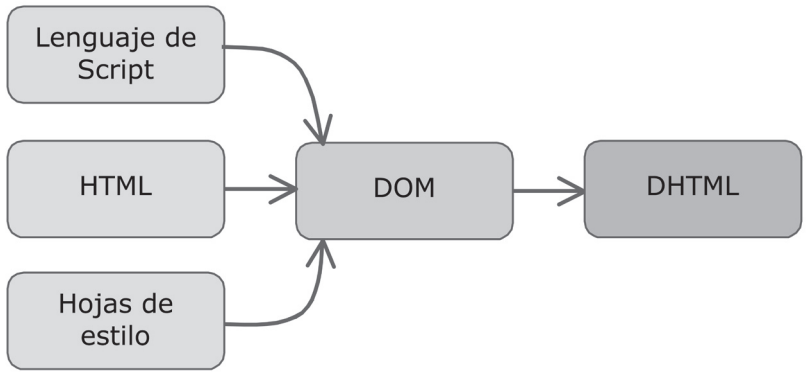


Figura 2.1. Ejemplo de solicitud HTTP.

La problemática que puede surgir del uso de DHTML es que al ser una agrupación de tecnologías dependientes del navegador, se pueden encontrar problemas de interpretación en función del navegador utilizado.

2.2.1. Hojas de estilo en cascada: CSS

CSS es el acrónimo de *Cascading Style Sheets* u hojas de estilo en cascada. Se trata de un lenguaje de definición de estilo para la presentación de documentos escritos con lenguajes de marcado (tipo XML). La aparición de esta tecnología ha permitido separar los estilos de presentación de un documento Web permitiendo dar un mayor dinamismo a los sitios Web. Con este lenguaje podemos definir los colores, fuentes y otros aspectos de la presentación.

2.2.1.1. Introducción de hojas de estilo en documentos HTML

Existen tres formas de dar estilo mediante el lenguaje CSS a un documento HTML.

- Mediante un archivo externo (**.css*) que se enlace al documento HTML

```
h1 {color: blue; text-align: center}
```

Listado 2.7. Ejemplo de archivo CSS.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
    <LINK rel="stylesheet" type="text/css"
    href="http://www.dtic.ua.es/grupoM/presentacion.
css"/>
  </HEAD>
  <BODY>
    <h1>Hola mundo azul</h1>
  </BODY>
</HTML>
```

Listado 2.8. Ejemplo de referencia de archivo CSS en HTML.

- Mediante el uso del elemento `<style>` dentro del documento HTML

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
    <STYLE type="text/css">
    h1 {color: blue; text-align: center}
    </STYLE>
  </HEAD>
  <BODY>
    <h1>Hola mundo azul</h1>
  </BODY>
</HTML>
```

Listado 2.9. Ejemplo de estilo mediante el uso de <style>.

- Mediante el uso del atributo `style` dentro de los elementos HTML

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <h1 style="color: blue">Hola mundo azul</h1>
  </BODY>
</HTML>
```

Listado 2.10. Ejemplo de estilo mediante el atributo style.

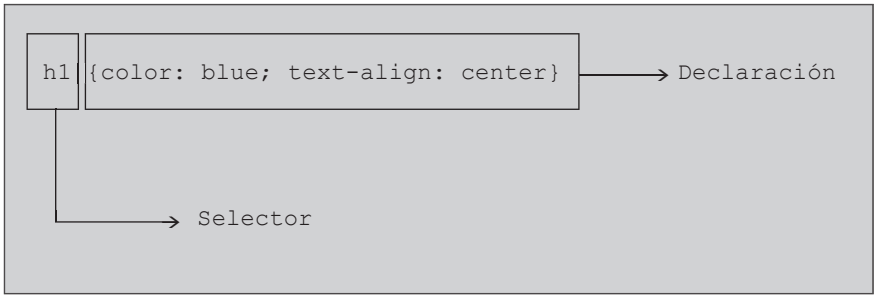
Este último caso es poco recomendable puesto que no permite una separación entre contenido y presentación.

2.2.1.2. Funcionamiento de las hojas de estilo CSS

CSS funciona a través de la declaración de reglas que establecen los estilos sobre determinados elementos. Un documento de estilos puede con-

tener una o más reglas que se aplicarán a determinados documentos HTML o XML. Las reglas se componen de dos partes:

- Selector que identifica la regla. Normalmente se refiere a la etiqueta o elemento HTML al que queremos dar el estilo. En otras ocasiones, es un nombre lógico único el cual es especificado en el atributo class de un elemento o etiqueta HTML al que queremos darle un estilo determinado. Esta última forma de uso es utilizada cuando diferentes elementos HTML tienen el mismo estilo. Además, en el primer caso podemos establecer varios estilos para un mismo elemento a través del atributo `class`.
- Declaración que especifica los estilos determinados para un elemento HTML.



Listado 2.11. Declaración de estilos.

En el primer ejemplo todos los elementos del documento HTML, `h1`, establecerían el texto entre dichas etiquetas a color azul y centrado.

Si yo quiero establecer varios estilos para el elemento `h1` se debería utilizar el nombre del elemento `h1` seguido por un `'.'` y un nombre lógico para cada regla. En el documento HTML, en cada elemento `h1` se debería utilizar el atributo class indicando el identificador del estilo a aplicar.

```
h1.primero {color: blue; text-align: center}
h1.segundo {color: red; text-align: center}
```

Listado 2.12. Ejemplo de archivo CSS.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo JavaScript</TITLE>
  </HEAD>
  <BODY>
    <h1 class="primero">Hola mundo azul</h1>
    <h1 class="segundo">Hola mundo rojo</h1>
  </BODY>
</HTML>
```

Listado 2.13. Uso de estilos en HTML.

2.2.2. El modelo de objetos de documento: DOM

El Modelo de Objetos de Documento (*Document Object Model*) define una manera de representar los elementos de un documento estructurado, tal como HTML o XML, como objetos con sus métodos y propiedades y como se accede y manipula el documento a través de sus objetos. Esta tecnología presenta una manera de acceder a los elementos de un documento de este tipo, tanto a sus elementos como a sus atributos, permitiendo añadir, eliminar o modificar estos elementos. Por lo tanto se puede definir como una interfaz de programación de aplicaciones (API) para documentos HTML o XML.

Con el Modelo de Objetos de Documento los programadores pueden construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido. Se puede acceder a cualquier cosa que se encuentre en un documento HTML o XML, y se puede modificar, eliminar o añadir usando el Modelo de Objetos de Documento, salvo algunas excepciones. En particular, aún no se ha especificado las interfaces DOM para los subconjuntos internos y externos de XML.

Los documentos DOM tienen una estructura jerárquica similar a una estructura de árbol pero no tiene porque mantener necesariamente esta organización. Puede además estar compuesto de varios documentos con este tipo de estructura.

```
<HTML>
  <HEAD>
    <TITLE>Ejemplo</TITLE>
  </HEAD>
  <BODY>
    <TABLE>
      <TR>
      <TD>Nombre</TD>
        <TD>Apellidos</TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

Listado 2.14. Documento XML.

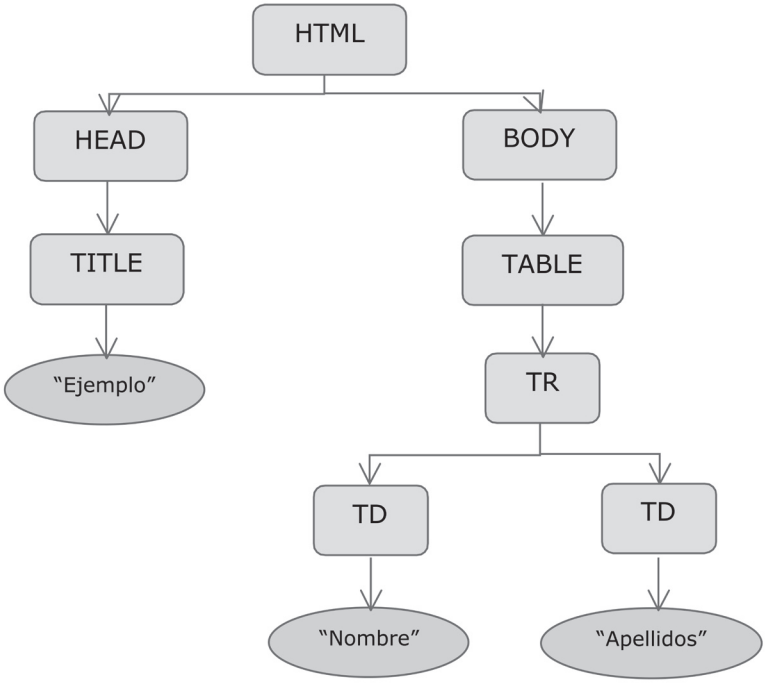


Figura 2.2. Representación estructurada del documento XML.

2.3. JAVASCRIPT Y XML ASÍNCRONO: AJAX

AJAX es el acrónimo de *Asynchronous JavaScript And XML*. Al igual que DHTML no se trata de ningún lenguaje de programación. AJAX es una técnica de desarrollo de aplicaciones Web que combina un conjunto de tecnologías con el objetivo de proveer al cliente una navegación ágil y rápida y convirtiendo el navegador en un entorno muy dinámico. Entonces, ¿Cuál es la diferencia con DHTML? Básicamente, se puede decir que AJAX es una combinación de DHTML junto con otra tecnología que es la que marca la principal diferencia, el *objeto XMLHttprequest* (este elemento tampoco es nuevo y fue propuesto por Microsoft en 1998). La verdadera diferencia de esta técnica radica en su filosofía de funcionamiento, dónde de forma transparente al usuario, el cliente mantiene una comunicación asíncrona con el servidor en un segundo plano. Realiza peticiones GET y POST obteniendo un documento XML como resultado y utilizando DOM le permite leer los datos y modificar la página. A efectos del usuario se verá que la página cambia de aspecto sin la necesidad de recargarla.

Las tecnologías básicas usadas para el desarrollo de aplicaciones Web con AJAX son:

- *XHTML y CSS* como estándares de presentación.
- *DOM* para mostrar e interactuar con la información.
- *XML y XSLT* manipulación e intercambio de datos (en la parte del servidor).
- *XMLHttpRequest* para el envío y la recepción de información de forma asíncrona.
- *JavaScript* como enlace para gestionar todas las tecnologías anteriores.

2.3.1. Modelo de aplicaciones Web con AJAX

Tradicionalmente, en las aplicaciones Web el usuario interactúa con el servidor a través de un formulario, que una vez completado, se le envía por medio del navegador como una petición Web. Este tipo de aplicaciones tienen ciertos límites y no es posible crearlas con las mismas características que las aplicaciones de escritorio (versatilidad, interactivas, etc.). Cada vez que se requieran datos del servidor se debe recargar elementos comunes (información de presentación). Esto hace, por un lado, que la transferencia de información sea mayor y por otro, que el usuario tenga que estar a la espera de cada nueva recarga.

Fundamentalmente, AJAX establece una capa entre la interfaz del cliente y la aplicación de servidor, que hace que las operaciones de comunicación y trasiego de información junto con el refresco de datos de cara al usuario se hagan de manera transparente para éste. De esta forma se consigue parte de la potencia de las aplicaciones de escritorio y además se mejora la comunicación, puesto que la nueva capa introducida (*motor AJAX*) solamente obtiene los datos necesarios en cada momento sin necesidad de descargar la estructura del documento en cada operación.

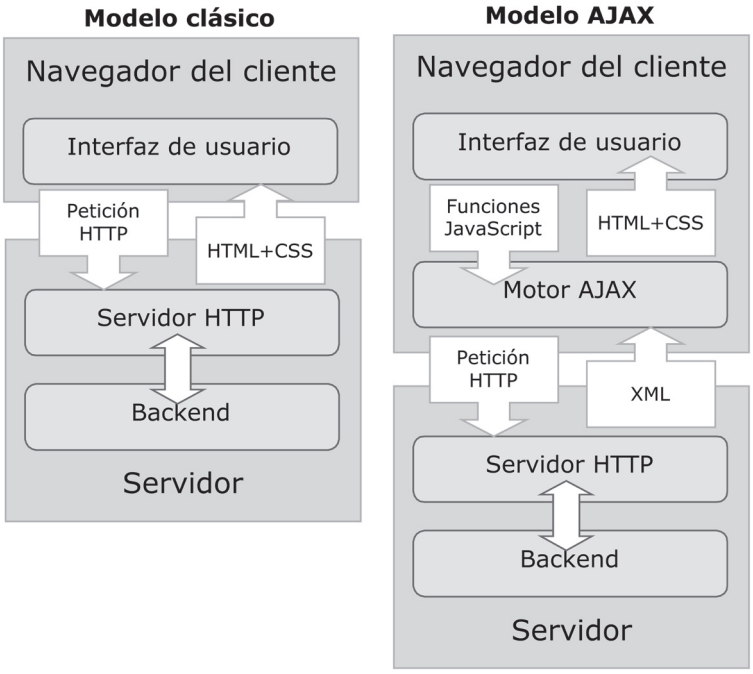


Figura 2.3. Modelo tradicional Web y modelo AJAX.

El *motor AJAX* o, conceptualmente, la nueva capa que se introduce, no es más que un conjunto de archivos *js* utilizados como librerías, que se encargan de *renderizar* la interfaz del usuario y realizar la comunicación con el servidor de manera transparente al usuario. Este motor permite establecer una interacción entre el usuario y la aplicación de forma asíncrona (figura 2.4), es decir, independientemente de la comunicación con el servidor.

Cualquier acción realizada por el usuario, en lugar de generar una petición HTTP, genera una llamada a una función JavaScript al *motor AJAX*. Si para llevarse a cabo la petición del usuario no es necesaria información del servidor o no requiere realizar una acción en el servidor, se resolverá en el propio *motor AJAX* ubicado en el cliente (validación de datos, visualización de información en memoria, etc.). Si por lo contrario, la petición requiere de la ejecución de un proceso en el servidor para generar la respuesta, el *motor AJAX* realiza las peticiones necesarias de manera asíncrona, mediante algún objeto como el *XMLHttpRequest*, sin parar la interacción con el usuario.

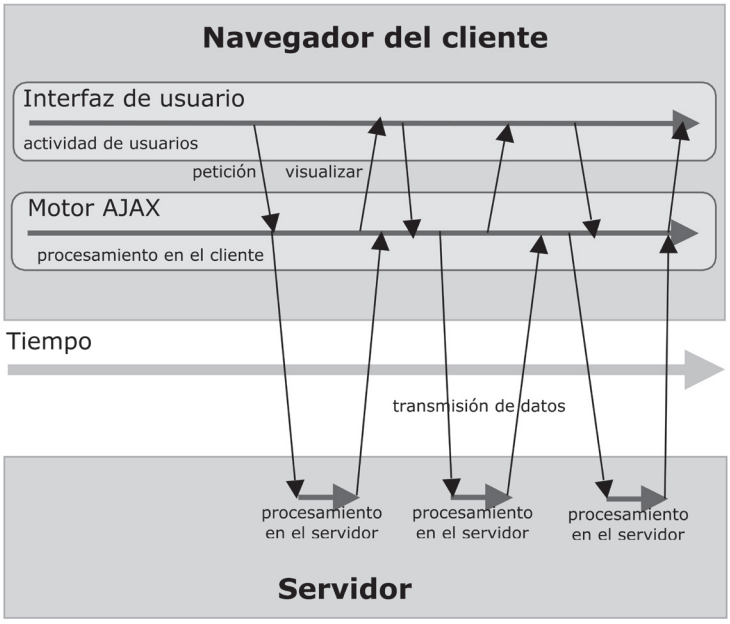


Figura 2.4. Modelo asíncrono de AJAX.

De una manera más concreta podríamos describir el funcionamiento de AJAX con los siguientes pasos:

- El usuario interactúa con la interfaz Web provocando un evento.
- El motor AJAX crea e inicializa un objeto *XMLHttpRequest*.
- El objeto *XMLHttpRequest* realiza una petición HTTP al servidor.

- El servidor procesa la petición y devuelve como resultado la información estructurada en XML.
- El objeto *XMLHttpRequest* obtiene la información como XML y se llama a la función encargada de procesarla.
- Se actualiza el DOM de la página asociada a la petición a partir del resultado devuelto por el servidor.

2.3.2. El objeto XMLHttpRequest

El objeto *XMLHttpRequest* fue desarrollado originalmente por Microsoft como un objeto ActiveX y está presente desde la versión 5 del Internet Explorer. Este objeto ha sido implementado en otros navegadores como en Mozilla desde la versión 1.0 y en Safari desde la 1.2. Por este motivo su uso se ha extendido a navegadores ampliamente utilizados. Este objeto ofrece un API que permite realzar conexiones HTTP con el servidor el cual puede devolver la respuesta usando XML, texto plano, JavaScript o JSON. El objeto ha sido utilizado por los diferentes *lenguajes de Script* y utiliza para la comunicación un canal de conexión independiente. Este objeto permite a lenguajes como JavaScript realizar peticiones HTTP a un servidor remoto sin la necesidad de volver a cargar la página que se este visualizando. Como se comentó anteriormente con este objeto se pueden estar haciendo peticiones y recibiendo respuestas del servidor en segundo plano sin que el usuario final sea consciente de este proceso.

Un pequeño problema es, que, mientras que para utilizar el objeto desde JavaScript en un navegador IE se instancia un objeto ActiveX como se muestra en la listado 2.15, en Mozilla o Safari se trata de un objeto nativo (listado 2.16), y su forma de instanciarlo es completamente diferente. Se debe tener en cuenta este aspecto para hacer la aplicación compatible con los navegadores.

```
Var req = new ActiveXObject("Microsoft.XMLHTTP");
```

Listado 2.15. Instancia del objeto como ActiveX.

```
Var req = new XMLHttpRequest();
```

Listado 2.16. Instancia del objeto nativo de Mozilla o Safari.

Con motivo de esta inconsistencia en el objeto es posibles establecer la funcionalidad en JavaScript que permita indicar el objeto que se debe usar, como se muestra en el listado 2.17.

```
var req;

function loadXMLDoc(url)
{
    // Opción para objeto XMLHttpRequest nativo
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
        req.onreadystatechange = processReqChange;
        req.open("GET", url, true);
        req.send(null);

    // Opción para objeto ActiveX de IE/Windows
    } else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
        if (req) {
            req.onreadystatechange = processReqChange;
            req.open("GET", url, true);
            req.send();
        }
    }
}
```

Listado 2.17. Instancia del objeto nativo de Mozilla o Safari.

Los métodos que ofrece el API *XMLHttpRequest* son:

- *abort()*: Detiene la petición actual.
- *getAllResponseHeaders()*: Devuelve todas las cabeceras de la respuesta como pares de etiqueta y valores en una cadena.
- *getResponseHeader("headerLabel")*: Devuelve el valor de una cabecera determinada.
- *open("method", "URL"[, asyncFlag[, "userName"[, "password"]]]):* Asigna la *URL* de destino, el método y otros parámetros opcionales de una petición pendiente.

- *send(content)*: Envía la petición, opcionalmente se puede enviar una cadena de texto o un objeto DOM.
- *setRequestHeader("label", "value")*: Asigna un valor al par *label/value* para la cabecera enviada.

De la misma forma las propiedades que ofrece son:

- *onreadystatechange*: El manejador del evento llamado en cada cambio de estado del objeto. Permite preparar la recepción de la respuesta del servidor asociando el manejador a una función que tratará la respuesta.
- *readyState*: Entero que indica el estado del objeto (0 = sin inicializar, 1 = cargando, 2 = fin de la carga, 3 = actualizando la información recibida, 4 = Operación completada).
- *responseText*: Cadena de texto con los datos devueltos por el servidor.
- *responseXML*: Objeto DOM devuelto por el servidor.
- *status*: Código numérico devuelto por el servidor, ejemplos: *404* si la página no se encuentra, *200* si todo ha ido bien, etc.
- *statusText*: Mensaje que acompaña al código de estado.

A continuación se muestra un pequeño ejemplo básico de uso del AJAX con la aplicación `"Hola Mundo"`.

Lo primero que se debe hacer es crear una variable (*httpRequest*) que será la que contenga la instancia al *objeto XMLHttpRequest*. Después, como se comentó anteriormente se debe crear la parte de código que nos permita instanciar el *objeto XMLHttpRequest* en función del navegador desde el cual se solicite la página.

Una vez instanciado se debe asociar al manejador la función que realizará la lectura de los datos enviados por el servidor. Este proceso como se muestra en el listado 2.19 se realiza con el atributo o propiedad `onreadystatechange` y la función se ejecutará cada vez que la propiedad `readyState` del objeto `XMLHttpRequest` cambie de estado. Para leer estos datos se deberá comprobar que la petición se encuentre en estado 4, como se realiza en la función asociada `alertContent`. En el ejemplo simplemente se mostraría un mensaje de alerta con el contenido de la respuesta del servidor, pero en función del objetivo de la aplicación se podría manejar el resultado mediante DOM para presentar al usuario los datos finales.

```
<script type="text/javascript" language="javascript">
  function makeRequest(url) {
    var httpRequest;

    if (window.XMLHttpRequest) { // Mozilla, Safari, ...
       httpRequest = new XMLHttpRequest();
       if (httpRequest.overrideMimeType) {
          httpRequest.overrideMimeType('text/xml');
       }
    }
    else if (window.ActiveXObject) { // IE
         try {
            httpRequest = new ActiveXObject("Msxml2.XMLHTTP");
         }
         catch (e) {
            try {
             httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch (e) {}
         }
    }
    if (!httpRequest) {
         alert('Giving up :( Cannot create an XMLHTTP instance');
         return false;
    }
```

Listado 2.18. "Hola Mundo" con AJAX, parte I.

Una vez asociada la función al manejador se debe abrir la conexión con el servidor mediante los métodos GET, POST o HEAD. Para ello se debe indicar la *URL* de la que se desea obtener los datos, el tipo de modo a usar (en AJAX siempre se usará el modo asíncrono indicando como valor del parámetro `true`). Para realizar la conexión se debe llamar al método `send` del *objeto XMLHttpRequest* después de `open`.

El método `send` envía la petición con los datos pasados como parámetros como cuerpo de la petición.

En la última parte del código de ejemplo del listado 2.19 se mostraría al usuario un enlace que al pulsarlo invocaría a la función que inicia todo el proceso (`makeRequest`).

```
    httpRequest.onreadystatechange = function() {
                                    alertContents(httpRequest);
                                        };
    httpRequest.open('GET', url, true);
    httpRequest.send(null);
  }
  function alertContents(httpRequest) {

    if (httpRequest.readyState == 4) {
        if (httpRequest.status == 200) {
                alert(httpRequest.responseText);
        }
        else {
             alert('There was a problem with the request.');
        }
    }
  }
</script>
<span
    style="cursor: pointer; text-decoration: underline"
    onclick="makeRequest('holamundo.xml')">
        Make a request
</span>
```

Listado 2.19. "Hola Mundo" con AJAX, parte II.

En el listado 2.20 se muestra el documento XML que devuelve el servidor como respuesta a la petición realizada por el cliente.

```
<?xml version="1.0" ?>
<root>
Hola mundo!!!
</root>
```

Listado 2.20. Archivo XML devuelto por el servidor (`holamundo.xml`).

2.4. APLICACIONES LIGERAS

Existe la posibilidad de extender el comportamiento y las funcionalidades de los navegadores Web. Por un lado, se pueden introducir componentes software para que se ejecuten o corran en el contexto de terceras

aplicaciones, en este caso en los navegadores Web. En este primer caso el navegador debe soportar este tipo de aplicaciones de forma nativa. Estas aplicaciones ligeras no tienen sentido de manera independiente. La programación de estos componentes es más compleja y requieren más conocimientos que el uso de los *lenguajes de Script*. Otra gran diferencia, es que estos componentes han sido compilados previamente a su utilización y lo que recibe el navegador Web es código binario. En el caso de los *Script,* es el propio navegador el que cada vez que se carga la página o se inicia una acción interpreta el código. Una de las ventajas de estos componentes es que suelen ser más potentes que los *lenguajes de Script* interpretados en el cliente puesto que se basan en lenguajes de programación de alto nivel con las características de dichos lenguajes. Pero no todos los navegadores soportan este tipo de aplicaciones de manera nativa, debido a la cantidad de aplicaciones ligeras que aparecen cada día. Por este motivo, la mayoría de los navegadores implementan una arquitectura de plug-ins o conectores, que permite añadir módulos para incorporar aplicaciones ligeras.

2.4.1. Applets

Un *Applet* es un programa escrito en lenguaje Java que se puede incluir en una página HTML de la misma manera que incluimos una imagen. Cuando se solicita desde un navegador una página Web, esta puede contener una referencia a un Applet. Cuando el navegador esté interpretando la página HTML llegará hasta la etiqueta que incluye el Applet y posteriormente le solicita al servidor la transmisión del código binario del Applet. Una vez el código es transferido al cliente, es posible ejecutarlo en el contexto de la máquina virtual de java asociada al navegador, pero en el lado del cliente (generalmente con la ayuda de un plug-in). Generalmente, los principales inconvenientes del uso de los Applets en las aplicaciones Web son: por un lado, el tiempo que se requiere para la transmisión del Applet y que dependerá de la línea que se posea y por otro lado, que el navegador se haya habilitado para soportar aplicaciones java, es decir que se incluya la máquina virtual de java para poder ejecutar el Applet.

Como desventajas, en relación con JavaScript, cabe señalar que los Applets son más lentos de procesar y que tienen espacio muy delimitado en la página donde se ejecutan, es decir, no se mezclan con todos los componentes de la página ni tienen acceso a ellos. Por este motivo, con los Applets de Java no se puede, directamente, hacer cosas como abrir venta-

nas secundarias, controlar Frames, formularios, capas, etc, pero si ampliar la funcionalidad de nuestro navegador e incluso acceder a componentes del lado del servidor.

La principal ventaja de utilizar Applets consiste en que son mucho menos dependientes del navegador que los S*cripts* en JavaScript, incluso independientes del sistema operativo del ordenador donde se ejecutan. Además, Java es más potente que JavaScript, por lo que las aplicaciones de los Applets podrán ser mayores.

En la figura 2.21 se puede ver el código para crear un sencillo Applet. Este tipo de Applet que extiende de la clase `java.applet.Applet` se utiliza cuando no vamos hacer uso de componentes *GUI* de tipo *Swing* y utiliza únicamente los componentes *AWT.* Pero esta opción comienza a estar en desuso y se utiliza la clase `javax.swing.JApplet` la cual permite introducir en el Applet componentes *GUI de tipo Swing*. El único problema con este último tipo de applets es que el plug-in instalado debe soportar *Swing*.

```
import java.applet.*;
import java.awt.*;
public class HolaMundo extends Applet {
    public void paint (Graphics g) {
           g.drawString ("¡¡Hola Mundo!!",10,80);
    }
}
```

Listado 2.21. Ejemplo de un Applet sencillo.

```
<html>
  <head>
    <title> Ejemplo simple de applet </title>
  </head>
  <body>
    <p>A continuación está la salida del programa</p>
    <applet code="Clock.class" width=170 height=150>
      <param name=bgcolor value="000000">
      <param name=fgcolor1 value="ff0000">
      <param name=fgcolor2 value="ff00ff">
    </applet>
    No hay disponible un intérprete de Java
</body>
</html>
```

Listado 2.22. Inserción de un Applet en un documento HTML.

2.4.1.1. Ciclo de vida de un Applet

Mediante la invocación de ciertos métodos, un navegador gestiona el ciclo de vida de un Applet, si el applet es cargado en una página Web.

El ciclo de vida de un Applet básicamente se compone de cuatro pasos relacionados con un método cada uno.

- El método `init` es utilizado para realizar los procesos de inicialización del Applet, si es necesario. Este método es invocado después de que el navegador lea el atributo `param` de la etiqueta `<applet>`.
- El método `start` es automáticamente invocado por el navegador después del método `init`. También se invoca a este método siempre que el usuario regrese a la página que lo contiene después de haber navegado por otras páginas.
- El método stop es invocado de forma automática siempre y cuando el usuario abandone la página que contiene el Applet.
- El método destroy es invocado cuando el navegador es cerrado siguiendo el cauce habitual.

De esta forma, el Applet puede ser inicializado una única vez, arrancado y parado una o más veces en su ciclo de vida y destruido, también, una única vez.

La etiqueta `<APPLET>` de arriba especifica que el navegador debería cargar la clase cuyo código compilado está en el fichero llamado `HelloWorld.class`. El navegador busca este fichero en el mismo directorio que contiene el documento HTML que contiene la etiqueta.

Cuando el navegador encuentra el fichero de la clase, la carga a través de la red, si es necesario, en el ordenador donde se está ejecutando el navegador. Entonces el navegador crea un ejemplar de la clase. Si incluimos un applet dos veces en la misma página, el navegador sólo cargará el Applet una vez y creará dos ejemplares de esa clase.

Cuando un navegador que soporte Java encuentra una etiqueta `<APPLET>`, reserva un área de pantalla de la anchura y altura especificadas, carga los *bytecodes* de la subclase Applet especificada, crea un ejemplar de la subclase y luego llama a los métodos `init` y `start` del ejemplar.

Incluimos los Applets en páginas HTML usando la etiqueta <`APPLET`>. Cuando un navegador visita una página que contiene un Applet suceden los siguientes pasos:

- El navegador busca el archivo *.class* de la subclase del Applet.
- La localización del archivo *.class* (que contiene los *bytecodes* Java) se especifica con los atributos `CODE` y `CODEBASE` de la etiqueta `<APPLET>`.
- El navegador descarga los *bytecodes* a través de la red hasta el ordenador del usuario.
- El navegador crea un ejemplar de la subclase Applet (cuando nos referimos a un Applet, generalmente nos referimos a este ejemplar).
- El navegador llama al método `init` del Applet (este método realiza cualquier inicialización que sea necesaria).
- El navegador llama al método `start` del Applet (este método normalmente arranca un *thread* que realiza las tareas del Applet).

Una subclase Applet es la clase principal, la clase controladora, pero los Applets también pueden usar otras clases. Estas otras clases pueden ser locales del navegador, proporcionadas como parte del entorno Java, o ser clases personalizadas que el usuario suministra. Cuando el Applet intenta utilizar una clase por primera vez, el navegador intenta encontrarla en el host en el que se está ejecutando. Si no puede encontrarla allí, busca la clase en el mismo lugar de donde vino la subclase de Applet. Cuando el navegador encuentra la clase, carga sus *bytecodes* (a través de la red, si es necesario) y continúa ejecutando el Applet.

Cargar código ejecutable a través de la red es un riesgo de seguridad clásico. Para los Applets Java, este riesgo se reduce porque el lenguaje Java está diseñado para ser seguro (por ejemplo, no permite punteros a memoria). Además, los navegadores compatibles Java mejoran la seguridad imponiendo sus propias restricciones. Estas restricciones incluyen no permitir a los Applets que carguen código escrito en otros lenguajes distintos de Java, y no permitiendo que los Applets lean o escriban ficheros en el *host* del navegador.

2.4.1.2. Instalación de Java plug-in en navegadores

Existen pequeños navegadores, como el *appletviewer* proporcionado por SUN en la máquina virtual de java JDK 1.2 que ya incorpora la compatibilidad con los Applets.

En el resto de navegadores independientes de Java, para poder visualizar un Applet en un navegador tipo IE o Mozilla Firefox es necesario

instalar el plug-in de Java. El *Java plug-in* es incluido como parte del entorno de ejecución de Java, JRE, y permite establecer una conexión entre los navegadores más populares y la plataforma Java. Este permite a los Applets ejecutarse en el contexto del navegador. Cargar el *Java plug-in* es un proceso muy sencillo que únicamente requiere instalar la máquina virtual de java JRE. Automáticamente se podrán visualizar los Applets en nuestro navegador.

Una vez instalado tenemos que asegurarnos que está habilitada en el navegador la opción para permitir usar el JRE y visualizar Applets.

En el navegador IE debemos ir al menú de *herramientas* → *opciones de Internet* → pestaña de *opciones avanzadas* y asegurarnos que en el *check box Java (Sun)* este marcado, de lo contrario no podremos visualizar el Applet.

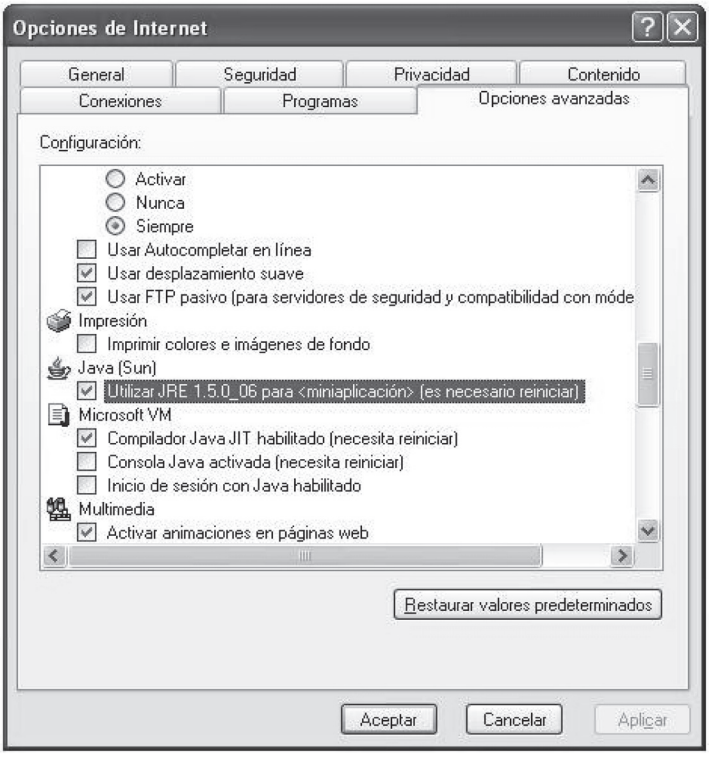


Figura 2.5. Habilitar en IIS el uso de Applets.

En el navegador Mozilla Firefox se requiere un proceso similar. En el menú *Herramientas* → *opciones* → *pestaña contenido* activamos la casilla cuya inscripción dice *Activar Java*. Con esto se obtiene el mismo resultado que en el proceso del IE.

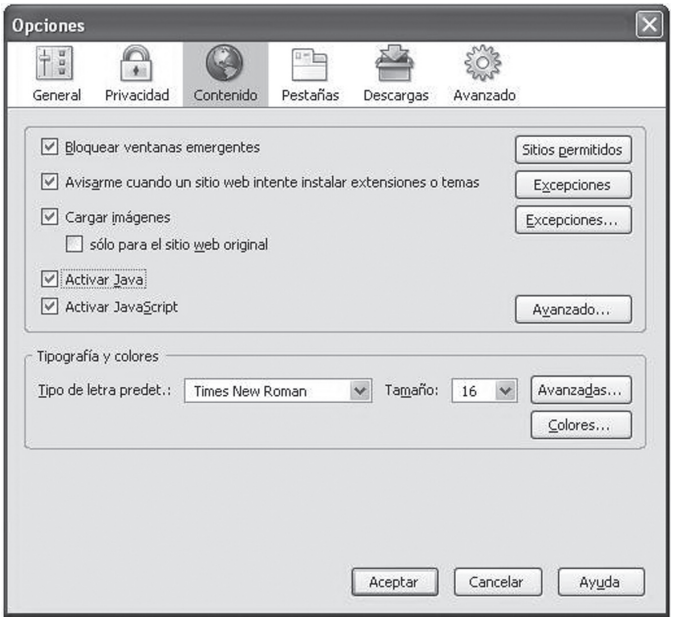


Figura 2.6. Habilitar en Firefox el uso de Applets.

El panel de control de java, el cual se puede encontrar en el panel de control del sistema operativo permite configurar y ver un amplio rango de parámetros de configuración para controlar las características de la máquina virtual de java en el propio entorno. Por tanto controlar también la configuración del plug-in. Para más información se puede acceder a la *URL* `http://java.sun.com/j2se/1.5.0/docs/guide/deployment/deployment-guide/jcp.html`.

2.4.2. ActiveX

Tecnología utilizada, entre otras cosas, para dotar a las páginas Web de mayores funcionalidades, como animaciones, vídeo, navegación tridimen-

sional, etc. Los controles ActiveX son pequeños programas que se incluyen dentro de estas páginas. Lamentablemente, por ser programas, pueden ser el objetivo de algún virus.

ActiveX es una tecnología de Microsoft para el desarrollo de páginas dinámicas. Tiene presencia en la programación del lado del servidor y del lado del cliente, aunque existan diferencias en el uso en cada uno de esos dos casos.

Son pequeños programas que se pueden incluir dentro de páginas Web y sirven para realizar acciones de diversa índole. Por ejemplo hay controles ActiveX para mostrar un calendario, para implementar un sistema de FTP, etc.

Son un poco parecidos a los Applets de Java en su funcionamiento, aunque una diferencia fundamental es la seguridad, pues un Applet de Java no podrá tomar privilegios para realizar acciones malignas (como borrarnos el disco duro) y los controles ActiveX sí que pueden otorgarse permisos para hacer cualquier cosa.

Los controles ActiveX son particulares de Internet Explorer aunque existen plug-ins específicos para poder ejecutar controles ActiveX dentro del contexto de otros navegadores.

Los controles ActiveX son componentes de software que se pueden descargar y que se utilizan para ampliar las funciones de Internet Explorer a través de elementos de la interfaz de usuario, como menús emergentes, botones.

Mientras se explora la Web se tiene protección ante la descarga de controles ActiveX no deseados, ya que si una página intenta descargar uno de ellos, Internet Explorer le mostrará un certificado firmado con el nombre de la compañía que creó el control. Este certificado aparece como un cuadro de diálogo de advertencia de seguridad. Si se encuentra en un sitio Web de su confianza (en este caso, Windows Update) y el control ActiveX lo proporciona una compañía en la que confía (en este caso, Microsoft), puede aceptar el certificado y permitir la instalación del control.

Aunque en principio los controles ActiveX fueron creados para utilizarse con Internet Explorer, posteriormente se han creado plug-ins para otros navegadores de uso común como Netscape y Mozilla Firefox.

Uno de los problemas a diferencia de los Applets es que los controles ActiveX tienen acceso completo al sistema operativo con el peligro que ello puede conllevar. Para controlar esta falla Microsoft creó un sistema de registro para que los navegadores puedan identificar y autenticar un control ActiveX antes de descargarlo y ejecutarlo.

```
<html>
  <head>
    <title> Ejemplo simple de activeX </title>
  </head>
  <body>
   <object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
width="160" height="144"
codebase="http://www.apple.com/qtactivex/qtplugin.cab">
      <param name="SRC" value="sample.mov">
      <param name="AUTOPLAY" value="true">
      <param name="CONTROLLER" value="false">
    </object>
  </body>
</html>
```

Listado 2.23. Inserción de ActiveX en un documento HTML.

En el listado 2.23. se muestra un ejemplo para incluir un control ActiveX dentro de un documento HTML (el control ActiveX para Quicktime). El atributo `classid` identifica de forma única que control ActiveX se va a usar. Un atributo `classid` con el valor `clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B` indica al navegador Internet Explorer que use el control ActiveX para Quicktime. El desarrollador debe conocer este valor. El atributo `codebase` indica la localización del control ActiveX. El usuario hará uso de este valor únicamente si no tienen instalado el control de forma que el navegador accederá a la ubicación y descargará el control. El navegador Internet Explorer automáticamente se ofrecerá para descargar e instalar el control.

2.4.2.1. Configuración de Internet Explorer con ActiveX

Ya se ha comentado anteriormente los peligros que puede conllevar permitir la ejecución de controles ActiveX sin control. En el navegador Internet Explorer por defecto las opciones de descarga y ejecución de ActiveX se encuentran deshabilitadas dejando el control al usuario si desea descargar y ejecutar los controles. En función del nivel de seguridad que tengamos asignado o definido se actuará de diversas formas.

La opción `personalizar`, da un mayor control a los usuarios avanzados y a los administradores sobre todas las opciones de seguridad. Por ejemplo, la opción `descargar` los controles ActiveX sin firmar se deshabilita de forma predeterminada en la *zona Intranet local* (la seguridad media es la

configuración predeterminada de la *zona Intranet local*). En este caso, Internet Explorer no puede ejecutar ningún control de ActiveX en la intranet de su organización porque la mayoría de las organizaciones no firman los controles ActiveX que sólo se utilizan internamente. Para que Internet Explorer ejecute los controles ActiveX sin firmar en la intranet de una organización, cambie en el nivel de seguridad de la *opción Descargar*, los controles ActiveX sin firmar a *Preguntar* o *Habilitar* para la *zona Intranet local*. En las opciones de seguridad en la opción *Nivel personalizado* se puede establecer el acceso a los archivos, controles ActiveX y secuencias de comandos.

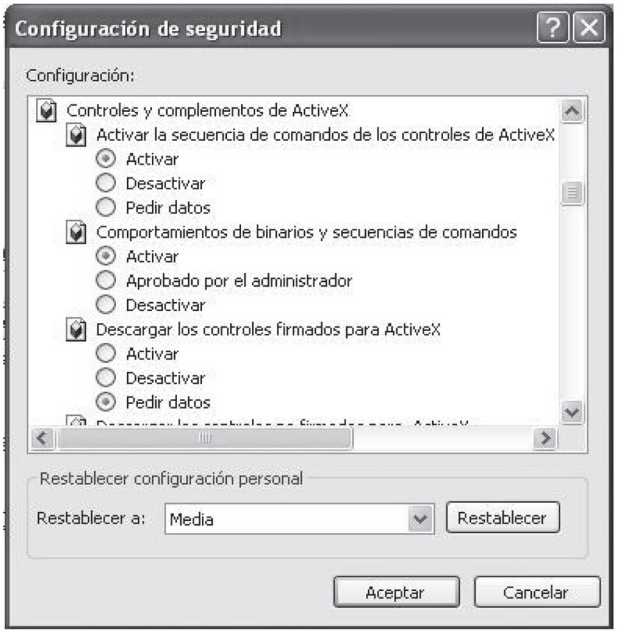


Figura 2.7. Opciones de seguridad para controles ActiveX en IE.

2.4.2.2. Plug-in de ActiveX para Mozilla

El navegador Mozilla no soporta controles ActiveX de forma nativa. Se ha desarrollado un plug-in, el cual también trabaja con Netscape 4.x, que permite utilizar controles ActiveX en estos navegadores. Este plug-in se ha creado con ciertas limitaciones puesto que no permite descargar

controles e instalarlos de forma automática. Solamente permite ejecutar controles que ya se encuentran instalados e identificados como seguro para ser incrustado.

Se puede obtener más información en el *Mozilla ActiveX Project* en la *URL* `http://www.iol.ie/~locka/mozilla/mozilla.htm`.

Se debe tener en cuenta que los controles ActiveX pueden suponer un riesgo para nuestro equipo.

A continuación se describe el proceso de instalación del plug-in para *Mozilla Firefox 1.5*. Se debe tener en cuenta que el plug-in se realiza para versiones específicas del navegador y que podría acarrear daños si se utiliza con versiones no adecuadas (se debería realizar una desinstalación del plug-in).

Se puede descargar el plug-in para la versión nombrada anteriormente en:

```
http://www.iol.ie/~locka/mozilla/plugin.htm#download
```

Para ello se debe descargar el archivo con extensión *xpi* y ejecutamos el navegador Firefox. En el menú *archivo* elegimos *abrir archivo* y seleccionamos el archivo *xpi* descargado (en este caso *mozactivex-ff-15.xpi*). Una vez seleccionado y abierto se no ofrece la opción de *instalar el plug-in*.

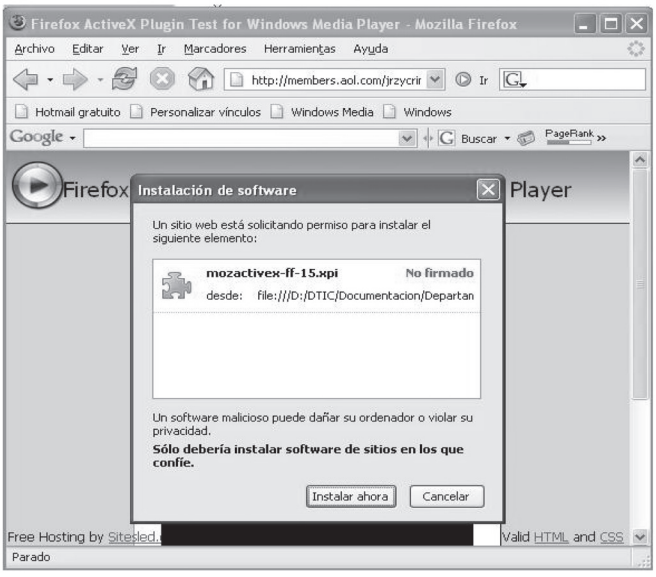


Figura 2.8. Opciones de seguridad para controles ActiveX en Mozilla.

Una vez instalado debemos cerrar el navegador y volverlo a ejecutar. Si no ha habido ningún problema se pueden visualizar controles ActiveX en este navegador.

Se puede realizar una prueba accediendo a la *URL* `http://members.aol.com/jrzycrim01/mozilla/wmp/wmpaxtest.html` y confirmar si se pueden ver los controles ActiveX que instancia el documento HTML descargado.

Para desinstalar el plug-in se deben seguir los siguientes pasos:

- Cerrar todas las instancias del navegador.
- Ir al directorio del *Mozilla Firefox*.
- Ir al directorio *Mozilla Firefox\plugins* y borrar el archivo `npmozax.dll`.
- Volver al directorio del *Mozilla Firefox* e ir al directorio *\components* para borrar los archivos `nsIMozAxPlugin.xpt` y `nsAxSecurityPolicy.js`.
- Acceder a la ruta *Mozilla Firefox\defaults\pref* y borrar el archivo `activex.js`.

2.4.3. Extensión del navegador mediante plug-ins

Los plug-ins son aplicaciones o módulos externos al navegador que permiten extender su funcionalidad para poder ejecutar mini-aplicaciones que doten de mayor potencia al navegador sin que el navegador tenga que soportarlas de una forma nativa.

Los plug-ins van a permitir que el navegador actúe como contenedor de otras aplicaciones gestionando su ciclo de vida. Esta técnica surge ante la imposibilidad de que un navegador Web sea capaz de procesar todos los tipos de datos que puede enviar como respuesta un servidor Web.

Un plug-in es cargado por el navegador si los datos enviados por el servidor son del tipo asociado al plug-in (la asociación se realiza mediante el tipo MIME).

2.4.3.1. Funcionamiento de los plug-ins

Se ha dividido la forma de actuar del navegador en referencia a los plug-ins en dos categorías. La primera en la cual se recibe un flujo de datos el cual requiere de la actuación de un plug-in directamente y no está insertado en código HTML. La segunda consiste en la inserción de objetos que requieren el uso de un plug-in por parte del navegador en código HTML.

En el primer caso, el cliente hace una petición mediante una *URL* a un servidor Web (`http://localhost/prueba.pdf`).

El servidor responde al cliente con el contenido mostrado en el listado 2.24. Cuando el navegador recibe el flujo de datos analiza la cabecera (Content-Type) y dependiendo del tipo MIME que se indica el navegador invocará al plug-in correspondiente para mostrar el contenido. En este caso se ejecutaría el plug-in de *Adobe Acrobat Reader*.

```
HTTP/1.1 200 OK
Date: Mon, 30 Oct 2006 20:17:17 GMT
Server: Apache/2.2.3 (Win32)
Last-Modified: Mon, 30 Oct 2006 19:52:12 GMT
ETag: "132f6-16e3-9b217560"
Accept-Ranges: bytes
Content-Length: 5859
Connection: close
Content-Type: application/pdf

%ÔÒ€Ë1.4 /L 5859/O 8/E 1750/N 1/T 5693/H [ 476 149]>>
xref
6 9
0000000016 00000 n
0000000625 00000 n
0000000701 00000 n
0000000833 00000 n
0000000916 00000 n
trailer
----8CFDA75A284D5A8033E016C87CBCE897—
......
......
```

Listado 2.24. Contenido de un archivo *pdf*.

El segundo caso se produce cuando se insertan objetos dentro de un documento HTML. En la especificación de HTML 4.01 se ha definido el elemento `Object` como mecanismo para invocar a los plug-ins. Este elemento es usado de forma diferente en el navegador Internet Explorer que en los navegadores basados en Mozilla. Para conocer más sobre este elemento se puede consultar la especificación de HTML 4.01 en `http://www.w3.org/TR/1999/PR-html40-19990824/`.

En primer lugar, el navegador Internet Explorer invoca a un plug-in creado como ActiveX. Esto implica que se debe indicar el identificador del ActiveX dentro del elemento `object` (atributo `classid`). Esto implica

```
<!ELEMENT OBJECT - - (PARAM | %flow;)*

<!ATTLIST OBJECT
%attrs;
declare     (declare)      #IMPLIED
classid     %URI;          #IMPLIED
codebase    %URI;          #IMPLIED
data        %URI;          #IMPLIED
type        %ContentType;  #IMPLIED
codetype    %ContentType;  #IMPLIED
archive     %URI;          #IMPLIED
standby     %Text;         #IMPLIED
height      %Length;       #IMPLIED
width       %Length;       #IMPLIED
usemap      %URI;          #IMPLIED
name        CDATA          #IMPLIED
tabindex    NUMBER         #IMPLIED>
```

Listado 2.25. DTD del elemento object.

que el programador debe conocer el identificador único de cada plug-in. El atributo codebase usado apunta a la localización donde está el archivo *CAB* que contiene el control del ActiveX que actúa como plug-in. En este contexto, el atributo codebase se usa como *mecanismo de obtención,* lo cual quiere decir, que se trata de una forma de obtener el controlador si no esta presente. Por ejemplo, si el control de ActiveX de Flash no está instalado, IE irá entonces a la *URL* indicada en el atributo `codebase` y obtendrá el control de ActiveX que permita visualizar la película.

Los atributos `param` especifican los parámetros de configuración para el plug-in.

Los navegadores basados en Mozilla soportan la arquitectura de plug-in de Netscape, los cuales no están basados en COM como el ActiveX (y

```
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=5,0,0,0"
width="366" height="142" id="myFlash">
    <param name="movie" value="javascript-to-flash.swf" />
    <param name="quality" value="high" />
    <param name="swliveconnect" value="true" />
</object>
```

Listado 2.26. Plugins en Internet Explorer.

por ello, no son llamados vía identificador único) si no, basados en el tipo MIME. Cuando el navegador interpreta un documento HTML y se encuentra con un elemento `object` este hace uso del atributo `type` el cual contiene el tipo MIME que identifica el tipo de objeto y por tanto le indica al navegador el plug-in al cual debe invocar. Puede suceder que el plug-in no esté instalado y se nos ofrezca un mensaje indicando la ubicación para descargarlo o que el propio navegador la conozca e indique si la queremos instalar.

```
<object type="application/x-shockwave-flash" data="javascript-to-
flash.swf"
width="366" height="142" id="myFlash">
  <param name="movie" value="javascript-to-flash.swf" />
  <param name="quality" value="high" />
  <param name="swliveconnect" value="true" />
  <p>You need Flash -- get the latest version from
  <a href= "http://www.macromedia.com/downloads/">here.</a></p>
</object>
```

Listado 2.27. Plug-ins en Firefox.

Para usar el plug-in de java en los navegadores e introducir Applets en los documentos HTML se sigue utilizando en la mayoría de las aplicaciones la etiqueta `<APPLET>` en lugar de `<OBJECT>` aunque en la especificación de HTML 4.1 se ha desechado esta opción y se recomienda usar el elemento `Object`.

2.5. MANTENIMIENTO DE LA SESIÓN: COOKIES

Puesto que HTTP es un *protocolo petición/respuesta* la peticiones son tratadas de manera individual. Las aplicaciones Web necesitan un mecanismo que les permita identificar un determinado cliente y el estado de cualquier conversación que se mantiene con los clientes. Una sesión es una corta secuencia de peticiones de servicio realizadas por un único usuario por medio de un único cliente para acceder al servidor. El estado de la sesión es la información mantenida en la sesión a través de las peticiones.

Una de las técnicas para el almacenamiento de esta información ha sido las denominadas *Cookies*. Se trata de una potente herramienta empleada por los servidores Web para almacenar y recuperar información acerca de sus visitantes.

Las *Cookies* son pequeñas porciones de información que se almacena en el disco duro del visitante de una página web a través de su navegador, a petición del servidor de la página. Esta información puede ser luego recuperada por el servidor en posteriores visitas. Al ser el protocolo HTTP incapaz de mantener información por sí mismo, para que se pueda conservar información entre una página vista y otra (como *login* de usuario, preferencias de colores, etc.), ésta debe ser almacenada, ya sea en la *URL* de la página, en el propio servidor, o en una *Cookie* en el ordenador del visitante.

Una *Cookie* no es más que un archivo de texto que algunos servidores piden a nuestro navegador que escriba en nuestro disco duro, con información acerca de lo que hemos estado haciendo por sus páginas.

Entre las mayores ventajas de las C*ookies* se cuenta el hecho de ser almacenadas en el disco duro del usuario, liberando así al servidor de una importante sobrecarga. Es el propio cliente el que almacena la información y quien se la devolverá posteriormente al servidor cuando éste la solicite. Además, las cookies poseen una fecha de caducidad, que puede oscilar desde el tiempo que dure la sesión hasta una fecha futura especificada, a partir de la cual dejan de ser operativas.

Entre las tareas que realiza una *Cookie* podemos destacar:

- Llevar el control de usuarios: cuando un usuario introduce su nombre de usuario y contraseña, se almacena una *Cookie* para que no tenga que estar introduciéndolas para cada página del servidor. Sin embargo, una *Cookie* no identifica a una persona, sino a una combinación de computador y navegador.
- Ofrecer opciones de diseño (colores, fondos, etc.) o de contenidos al visitante.
- Conseguir información sobre los hábitos de navegación del usuario, e intentos de *spyware*, por parte de agencias de publicidad y otros. Esto puede causar problemas de privacidad y es una de las razones por la que las *Cookies* tienen sus detractores.

2.5.1. Funcionamiento de las Cookies

El protocolo HTTP ha creado una serie de cabeceras que permite realizar operaciones con *Cookies* (crear, obtener *Cookies*, cambiar el tiempo de expiración, borrar, etc.).

Cuando un cliente realiza una petición a un servidor Web, este como respuesta puede enviar información en la cabecera del documento HTTP para la creación de *Cookies* en el cliente. Esta operación se realiza mediante la directiva de cabecera `Set-Cookie`.

Es el servidor el que inicia la sesión al responder al cliente con un mensaje que tiene una cabecera para establecer una *Cookie* (`Set-Cookie`). Cuando el cliente recibe esta respuesta, en la siguiente petición incorporará una cabecera para informar de las *Cookies* que tiene (`Cookie`). Ante esta petición puede tener en cuenta la *Cookie* o no para dar la respuesta, y para ello puede establecer el mismo valor u otro valor para la *Cookie* o no enviar ninguna `cookie`. Para acabar la sesión, el servidor debe establecer una *Cookie* con duración "0".

El servidor puede tener varias cabeceras para establecer varias *Cookies* en una misma respuesta, aunque si el mensaje pasa por un proxy o una pasarela, pueden convertirlo en una sola cabecera para establecer *Cookies*. Para establecer una *Cookie* se usa la cabecera `Set-Cookie`, cuya sintaxis es:

```
Set-Cookie: nombre=valor *[;modificador]
```

Con esto se establece que por lo menos ha de haber un par nombre/valor y cada par puede tener o no modificadores.

Los modificadores de un par nombre, valor pueden ser:

- `Comment = valor`, el servidor informa del uso de la *Cookie*.
- `Domain = valor`, indica el dominio en el que la *Cookie* es válida.
- `Max-Age = valor`, validez de la *Cookie* en segundos, después se descarta.
- `Path = valor`, indica el subconjunto de *URL* a las que afecta la *Cookie*.
- `Secure`, indica que el cliente debe usar en entornos seguros para contactar con el servidor.
- `Version = valor`, la versión de la especificación de mantenimiento de estado que es.

Se pueden enviar varias *Cookies* en una misma cabecera separando los pares nombre, valor con ",".

Además de enviar la *Cookie*, el servidor puede establecer las condiciones para que la *Cookie* se ponga en una *caché* o no.

Un ejemplo de la cabecera de una respuesta:

```
Set-Cookie: NOMBRE=VALOR; expires=FECHA; path=PATH;
domain=NOMBRE_DOMINIO; secure
Set-Cookie: ID=789; path=/
Set-Cookie: NOMBRE=Juan; path=/
……
<html>
……
</html>
```

Listado 2.28. Ejemplo de respuesta del servidor.

El cliente trata por separado las distintas informaciones de estado que le llegan por medio de respuestas que establecen *Cookies*. Además toma valores por defecto para los atributos que no están presentes de los pares nombre/valor.

Además el cliente puede rechazar una *Cookie* por razones de seguridad o violaciones de la privacidad. Para rechazar una *Cookie* tiene una serie de reglas que aplica en cada caso.

Si el cliente recibe una *Cookie* con un nombre que ya tiene asociado a otra *Cookie* y cuyos atributos `Domain` y `Path` son exactamente iguales, entonces la nueva *cookie* reemplaza a la anterior. Además si la nueva *Cookie* tiene el atributo `Max-Age` con un valor de 0, entonces elimina la *Cookie* en vez de reemplazarla.

Como el cliente tiene un espacio limitado para las *Cookies*, ha de ir eliminando *Cookies* antiguas para hacer sitio a las nuevas. Esta operación la puede realizar siguiendo algún algoritmo como el *LRU* (*Least Recently Used*), o *FIFO* (*First In First Out*).

Si una *Cookie* tiene el atributo `Comment`, entonces el cliente ha de almacenar el comentario para que el usuario (humano) pueda leerlo.

Cuando el cliente realiza una petición al servidor, lo hace por medio de una *URI*. Para cada petición, además del mensaje envía una cabecera `Cookie` para informar al servidor de las *cookies* que tiene y que afectan a esa petición. La sintaxis para esta cabecera es la siguiente:

```
Cookie: versión *((;|,)nombre = valor
                                  [;path] [;dominio])
```

Donde versión es:

```
Version = valor
```

path es:

```
Path = valor
```

Y dominio es:

```
Domain = valor
```

El valor de la versión es el del atributo Versión si alguna de las *Cookies* lo impone, si no es 0. El valor de `path` ha de ser el impuesto por las *cookies* si alguna lo establece, si no se omite. El caso del dominio se trata igual que el del `path`.

En la lista de *Cookies* aparecen las que satisfacen los criterios:

- El nombre del servidor ha de tener el mismo dominio que la *Cookie*.
- El `path` de la *Cookie* ha de ser un prefijo de la *URI* que representa la petición.
- El límite de tiempo de la *cookie* no ha sido sobrepasado.

Cuando el servidor recibe una petición con una cabecera *Cookie*, ha de interpretar los pares nombre/valor teniendo en cuenta que los nombres que comienzan por el símbolo `"$"` son especiales y hacen referencia a la *Cookie* anterior, y si no hay una *Cookie* anterior, hacen referencia a todas las *Cookies* de esta cabecera.

En la parte del cliente se ha establecido la cabecera *Cookie* que permite informar de las *Cookies* existentes.

En la cabecera de la siguiente solicitud:

```
NOMBRE1=VALOR1; NOMBRE2=VALOR2; …
Cookie: ID=789; NOMBRE=JuanBla, bla, ….
……
<html>
……
</html>
```

Listado 2.29. Ejemplo de petición del cliente.

2.5.2. Habilitación de las Cookies

La mayor parte de los navegadores modernos soportan *Cookies*. Sin embargo, un usuario puede normalmente elegir si las *Cookies* deberían ser utilizadas o no. A continuación, las opciones más comunes:

- Las *Cookies* no se aceptan nunca.
- El navegador pregunta al usuario si se debe aceptar cada *Cookies*.
- Las *Cookies* se aceptan siempre.

El navegador también puede incluir la posibilidad de especificar mejor qué *Cookies* tienen que ser aceptadas y cuáles no. En concreto, el usuario puede normalmente aceptar alguna de las siguientes opciones: rechazar las *Cookies* de determinados dominios; rechazar las *Cookies* de terceros; aceptar *Cookies* como no persistentes (se eliminan cuando el navegador se cierra); permitir al servidor crear *Cookies* para un dominio diferente. Además, los navegadores pueden también permitir a los usuarios ver y borrar *Cookies* individualmente.

La mayoría de los navegadores que soportan JavaScript permiten a los usuarios ver las *Cookies* que están activas en una determinada página escribiendo `javascript:alert("Cookies: "+document.cookie)` en el campo de dirección.

En el navegador Mozilla Firefox podemos configurar los parámetros para permitir el almacenamiento de *Cookies*. Accediendo al menú de *herramientas* → *opciones* → *pestaña de privacidad* y en la *sub-pestaña de cookies* podemos configurar todas las posibilidades.

Para configurar el almacenamiento de *Cookies* en el navegador Internet Explorer debemos realizar los siguientes pasos. Ir al menú de *herramientas* → *opciones de Internet* → *pestaña de privacidad* → *opciones avanzadas*. Ahora se pueden configurar las opciones de las *Cookies* en el cliente.

Si utilizamos en navegador Mozilla Firefox las *Cookies* son almacenadas en un único archivo llamado `cookies.txt` en `%HOME-PATH%\Datos de programa\Mozilla\Firefox\Profiles\hjpv0b0b.default.`

Las cookies en Windows, utilizando el navegador Internet Explorer, normalmente se almacenan en `%HOMEPATH%/cookies`. En este directorio se pueden encontrar un elevado número de archivos con la información mantenida con cada sesión que se ha establecido con el servidor.

IE ofrece también una interfaz menos completa que solo permite eliminar todas las *Cookies*.

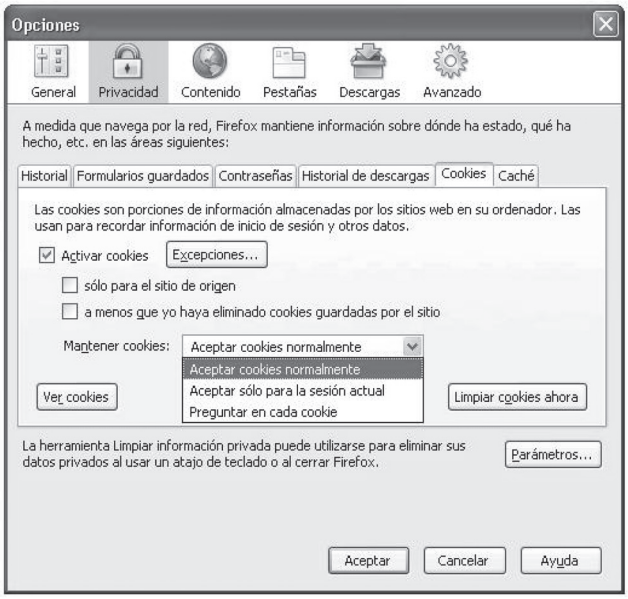


Figura 2.9. Habilitar en Firefox para el uso de *Cookies*.

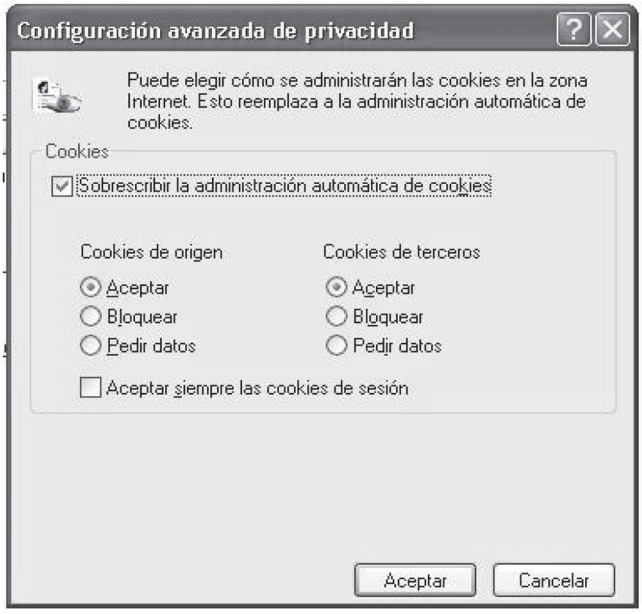


Figura 2.10. Habilitar en IE para el uso de *Cookies*.

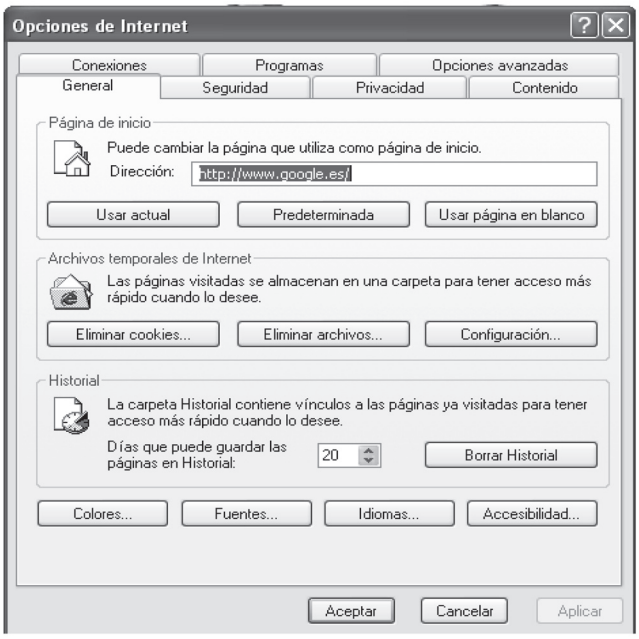


Figura 2.11. Interfaz que permite la eliminación de *Cookies*.

En ciertas ocasiones, por motivos de seguridad, no se permiten las *Cookies*, y por tanto, se debe acudir a otro tipo de técnicas como puede ser la reescritura de la *URL*. Esta técnica consiste en la modificación de la *URL* por parte del servidor para identificar cada *URL* con una sesión de un cliente.

3. AMPLIACIONES EN EL SERVIDOR

Si en el apartado anterior nos centrábamos en la posibilidad de personalizar nuestro cliente ligero engordándolo mediante diversas técnicas para dotar al servicio http de mayor funcionalidad, en este capítulo haremos lo propio con el servidor Web.

La mayor parte de las técnicas analizadas hasta el momento se basan en trasladar más o menos cantidad de conocimiento (*know-how*) desde el servidor hacia el cliente. Aunque esta técnica es muy interesante en numerosas ocasiones —sobre todo cuando se está buscando un control más fino de la interfaz y una respuesta más rápida al usuario—, en general resultan muy insuficientes para descargar al servidor de las sucesivas oleadas de demanda a la que debe enfrentarse a medida que se adentra en las turbulentas aguas del mundo de los negocios.

Las aplicaciones CGI nos permiten salir del paso, pero no están preparadas para soportar un gran número de transacciones, sobre todo porque el servidor no tiene el control de la ejecución de estas aplicaciones: no sabe realmente cómo están funcionando y no puede realizar optimizaciones de carga y ejecución.

Nuevamente, al no existir un estándar ampliamente reconocido, cada fabricante se lanzó a una carrera para proponer soluciones que pudieran lograr hacerse con la aceptación del mercado y, por lo tanto, convertirse con el tiempo en un estándar *de facto*. La realidad es que, aunque muchas ya han sido desechadas por este mercado, muchas otras se han afianzado en uno u otro contexto y, en la actualidad, conviven de una forma más o menos digna formando parte de las aplicaciones que operan sobre la Red.

3.1. EXTENSIONES DEL SERVIDOR

Una de las primeras ideas que se les ocurrió a los fabricantes de servidores Web consistió en permitir que sus clientes pudieran «construirse» su servidor «a medida». Mediante determinados lenguajes nativos de la plataforma y siguiendo determinadas especificaciones podremos realizar filtros o extensiones al servidor Web, a fin de dotarle de funcionalidades de las que en principio carece. Obsérvese que la intención es la misma que en el caso de la interfaz CGI, pero aquí lo que se va a construir es una librería de acceso dinámico (`.dll` en Windows, `.so` en sistemas Unix). Esto significa que, a diferencia de los programas CGI que se instancian cada vez que son llamados, estas librerías, una vez cargadas en memoria, van a permanecer allí para atender cualquier petición posterior que se requiera de ellas (figura 3.1). Y, lo que es más interesante, se ejecutan como un módulo más, perfectamente entrelazado, del propio servidor.

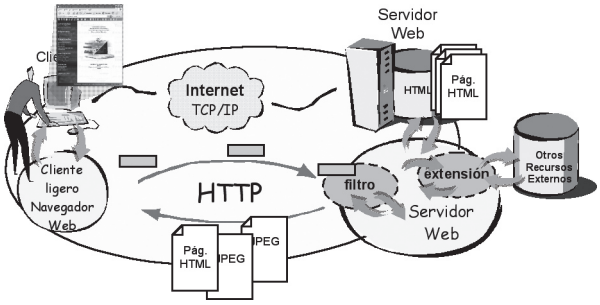


Figura 3.1. Escenario genérico para un modelo de filtros y extensiones del Servidor Web.

Aunque a nivel de desarrollo son equivalentes, funcionalmente un filtro se emplea para realizar un trabajo previo sobre la solicitud HTTP que llega al servidor Web mientras que las extensiones son invocadas (generalmente a través de estas mismas solicitudes HTTP) a posteriori para acceder de una forma eficiente a recursos o funcionalidades externas.

Para facilitar la labor, los diferentes fabricantes que han optado por esta técnica proporcionan una interfaz de programación de aplicaciones (API –*Application Programming Interface*) adecuada para su servidor Web. De esta forma, *Information Server API* (ISAPI) está orientado a

Internet Information Server (IIS) de Microsoft y *Netscape API* (NSAPI), a los servidores Web de Netscape (Enterprise Netscape Server).

Por tanto, mediante esta tecnología ganamos sustancialmente en rendimiento y aprovechamiento de los recursos de la máquina. Sin embargo, nos encontramos con el gran inconveniente de estar cerrada, no sólo a una determinada plataforma, sino también a un determinado servidor Web e, incluso, a una determinada versión del mismo.

Otro inconveniente que ha contribuido a que esta técnica no sea muy empleada en la práctica es que la programación de estas API es bastante compleja o, como mínimo, con una curva de aprendizaje mayor que la que se requiere para realizar un programa CGI mediante un lenguaje y técnicas más o menos familiares para los desarrolladores.

3.2. PÁGINAS ACTIVAS

Esta tecnología está basada en incrustar código escrito en un lenguaje tipo *script* dentro de las propias páginas HTML con el fin de generar páginas dinámicas. Para ello, el servidor Web interpretará este código, accederá, si es necesario, a los recursos externos que precise y generará dinámicamente código en lenguaje HTML antes de servir la página al cliente (figura 3.2).

Esta forma de desarrollar aplicaciones basadas en Web es de las más aceptadas actualmente ya que existe una amplia alternativa de lenguajes más o menos dependientes de la plataforma y a que, debido a su concepción —código script incrustado en HTML—, se puede emplear cualquier editor HTML como herramienta de desarrollo.

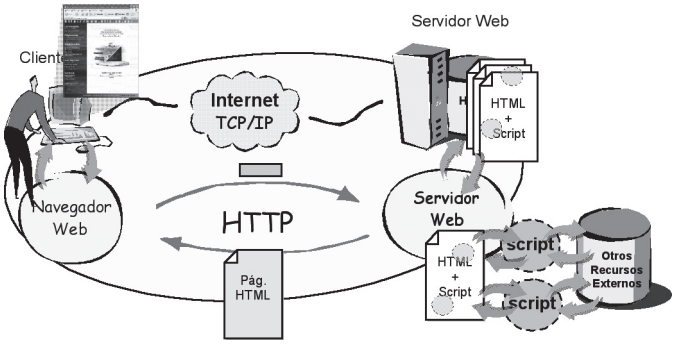


Figura 3.2. Escenario de desarrollo para una página activa.

A estas características que lo hacen popular, debemos unir otras de carácter más operativo, como el control que se obtiene entre la aplicación (el código script interpretado) y el servidor Web. Además, esta tecnología habilita el concepto de sesión de usuario, es decir, podemos conocer el estado de la conexión de un usuario en un momento determinado. Al mismo tiempo, gestiona bien los recursos externos, como es el caso de accesos a bases de datos, permite una gran flexibilidad para compartir los mismos y para la gestión de la petición y respuesta Web. Finalmente, cómo el código se interpreta y ejecuta en el servidor, se mantiene la independencia del cliente Web.

A cambio, el desarrollador deberá aprender el lenguaje script correspondiente, el cual es propietario del servidor Web que lo ha de interpretar. Según esto, podemos encontrar diferentes alternativas como son: PHP, ASP, *LiveWire* o JSP.

A continuación analizaremos muy brevemente cada una de estas propuestas y mostraremos un sencillo ejemplo de cómo se puede presentar una página de saludo equivalente en cada una de ellas y cuya representación gráfica puede observarse en la figura 3.3.

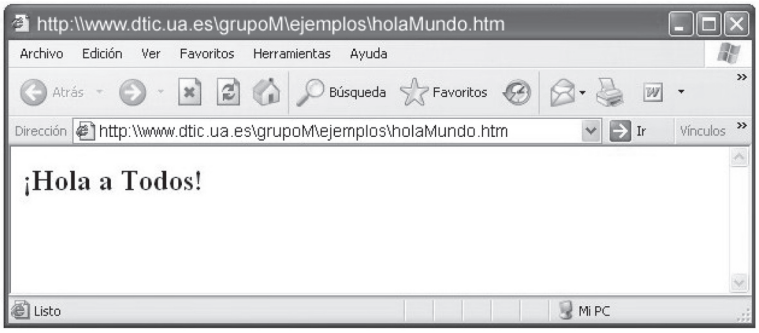


Figura 3.3. Representación por un navegador de la página HTML `holaMundo.htm`.

3.2.1. PHP

PHP es el acrónimo recursivo de «*PHP: Hypertext Preprocessor*» y puede considerarse como la propuesta pionera para el desarrollo de páginas dinámicas de una forma sistemática mediante el concepto de «páginas activas».

Puesto que esta característica no estaba incluida en los primeros servidores Web, se desarrolló mediante lenguaje PERL, como una serie de aplicaciones CGI que realizaban el preprocesamiento de páginas HTML con un código incrustado en etiquetas especiales al estilo de Perl. En la actualidad, diversos servidores Web, como Apache, ya incluyen esta característica de forma nativa.

La familiaridad de los desarrolladores en este tipo de lenguajes, la facilidad de uso y desarrollo, incluso, mediante un sencillo editor HTML, la potencia y el control que proporciona junto con su buena integración con el servidor Web, han contribuido notablemente a que esta tecnología sea plenamente aceptada por la comunidad Web para el desarrollo de sitios dinámicos y aplicaciones Web en general.

En el listado 3.1 puede observarse una sencilla página HTML denominada `holaMundo.php` con código PHP incrustado dentro de la etiqueta especial `<?php …?>`, cuya interpretación por un navegador Web convencional puede observarse en la figura 3.3.

```
<html>
<head>
</head>
<body>
<h2>
<?php echo "¡Hola a Todos!"; ?>
</h2>
</body>
</html>
```

Listado 3.1. Contenido de la *página activa* **holaMundo.htm** diseñada para mostrar en el navegador un mensaje de bienvenida generado dinámicamente mediante *php*.

3.2.2. LiveWire

Este lenguaje fue desarrollado originalmente por Netscape en 1995, con la finalidad de proporcionar un mayor dinamismo a las páginas Web.

Los servidores Web de *Netscape* e *Iplanet* incluyen LiveWire que representa un entorno de desarrollo que utiliza JavaScript ejecutado desde el servidor para crear aplicaciones (de manera similar a CGI, JSP y ASP).

A diferencia del código JavaScript que se ejecuta en el cliente (en algún *Browser* como Navigator o Explorer) las aplicaciones LiveWire JavaScript se ejecutan en el propio servidor. Para ello, primero son compiladas y almacenadas en un archivo ejecutable binario. Durante el proceso de ejecución, Livewire genera código HTML de manera dinámica. Por supuesto este código también puede incluir código Javascript para el navegador. Finalmente, el resultado es enviado al cliente a través del servidor Web para que éste interprete y muestre el resultado.

En el listado 3.2 puede observarse una sencilla página HTML denominada `holaMundo.htm` con código Livewire Javascript incrustado entre las etiqueta `<server>` y `</server>`, cuya interpretación por un navegador Web convencional puede apreciarse en la figura 3.3. Es muy importante no confundir Livewire javascript con el código javascript que se ejecuta en el cliente y que, aunque es el mismo, iría entre las etiquetas `<script>` y `</script>`.

```
<html>
<head>
</head>
<body>
<h2>
<SERVER>
    write("¡Hola a Todos!")
</SERVER>
</h2>
</body>
</html>
```

Listado 3.2. Contenido de la *página activa* **holaMundo.htm** que muestra en el navegador un mensaje de bienvenida generado dinámicamente mediante *Liveware Javascript*.

3.2.3. ASP

Los servidores Microsoft implementan la tecnología de páginas activas bajo el nombre de ASP (Active Server Pages). ASP se apoya en *Visual Basic Script* (VBScript) como lenguaje de *scripting* y está orientada al servidor *Internet Information Server* (IIS).

Las páginas pueden ser generadas incrustando código en las páginas HTML. Este código es interpretado por el servidor y desde el mismo se puede, como en el resto de casos, acceder a recursos y funcionalidades del equipo servidor o, incluso, de equipos remotos, incluyendo bases de datos.

En el listado 3.3 se muestra una sencilla página HTML denominada `holaMundo.asp` con código ASP incrustado dentro de la etiqueta especial `<% … %>`, cuya interpretación por un navegador Web convencional puede apreciarse en la figura 3.3.

```
<html>
<head>
</head>
<body>
<h2>
<% Response.Write "¡Hola a Todos!" %>
</h2>
</body>
</html>
```

Listado 3.3. Contenido de la *página activa* **holaMundo.asp** que muestra en el navegador un mensaje de bienvenida generado dinámicamente mediante *ASP*.

3.2.4. JSP

JavaServer Pages (JSP) es la propuesta desarrollada por Sun Microsystems para generar páginas Web de forma dinámica en el servidor. Esta tecnología está basada en Java y permite a los programadores generar dinámicamente HTML, XML o algún otro tipo de página Web. Para ello, como en el resto de casos, se inserta código, en este caso Java, dentro del contenido estático.

En versiones más recientes de la especificación se añadió la posibilidad de ejecutar *clases java* referenciadas desde la página HTML, mediante etiquetas denominadas «taglib». La asociación de las etiquetas con las clases java se declara en archivos de configuración escritos en XML.

La principal ventaja que presenta JSP frente a otras tecnologías es la posibilidad de integrarse con clases Java (`.class`). Esto permite organizar las aplicaciones Web en diferentes niveles, almacenando en clases java

las partes que consumen más recursos o las que requieren más seguridad, manteniendo únicamente la parte encargada de proporcionar formato al documento HTML dentro del archivo JSP. Además, Java se caracteriza por ser un lenguaje que puede ejecutarse en cualquier sistema que incorpore una máquina virtual de java (JVM). Teniendo en cuenta que en la actualidad se pueden encontrar implementaciones de JVM para casi cualquier plataforma, la portabilidad de las aplicaciones es realmente considerable.

Como ocurría en el caso de LiveWire, antes de que el servidor Web pueda ejecutar el código java, debe compilarlo, generando un objeto «servlet» (ver apartado 3.3.1). Aunque este proceso ralentiza inicialmente la ejecución, en posteriores invocaciones no sólo no se producirá retardo, sino que se ejecutará más rápido y podrá disponer del API de Java en su totalidad.

En el listado 3.4 se presenta el código fuente de una sencilla página HTML denominada `holaMundo.jsp` con código JSP incrustado dentro de la etiqueta especial `<% … %>`, cuyo resultado, interpretado por un navegador Web convencional, puede verse en la figura 3.3.

```
<html>
<head>
</head>
<body>
<h2>
<%= "¡Hola a Todos!" %>
</h2>
</body>
</html>
```

Listado 3.4. Contenido de la página activa **holaMundo.jsp** que muestra en el navegador un mensaje de bienvenida generado dinámicamente mediante JSP.

3.2.5. ASP .NET

ASP.NET es la plataforma unificada de Microsoft para el desarrollo Web que proporciona a los desarrolladores los servicios necesarios para crear aplicaciones Web empresariales.

Microsoft ha visto en la estrategia de separar la lógica de la aplicación del contenido estático y de los aspectos de presentación desarrollada por

SUN, su más directa competidora, una seria amenaza. Esta es la principal motivación que le ha llevado a incorporar en su plataforma *.NET* un lenguaje de *scripts ASP.NET* que permite ser integrado con clases *.NET* (ya estén desarrolladas en *C++*, *VisualBasic* o *C#*) del mismo modo que JSP se integra con *clases Java*.

Esta nueva tecnología se ha desarrollado como parte de la estrategia *.NET* para el desarrollo Web, con el objetivo de resolver las limitaciones de ASP y posibilitar la creación de *software como servicio*.

Para distinguir unas versiones ASP de otras, las versiones «pre-.NET» se denominan actualmente «ASP clásico».

3.3. COMPONENTES EN LA PARTE DEL SERVIDOR

Un componente en la parte del servidor o un «servicio ligero» conceptualmente no difiere mucho de los componentes en la parte del cliente o aplicaciones ligeras tratadas en el apartado 2.3. En cualquier caso, la diferencia fundamental radica en que este software ahora no se transfiere al cliente, sino que se ejecuta directamente en el propio servidor Web.

Según esta definición, los componentes en la parte del servidor también podrían confundirse con las aplicaciones CGI analizadas en el punto 1.3, sin embargo, también existe una diferencia notable con respecto a éstas y es que su ejecución sí se desarrolla dentro del contexto del propio servidor Web —recordemos que en el caso de las aplicaciones CGI, dicha ejecución era gestionada por el sistema operativo en un contexto diferente al del servidor Web—. Esto dota al servidor y a la propia aplicación de un control mutuo que permite superar las limitaciones del modelo CGI. Un servidor Web capaz de ejecutar este tipo de componentes se denomina también, «contenedor Web».

Funcionalmente, los componentes software en la parte del servidor se pueden considerar *piezas de software* creadas para encapsular una determinada *lógica de negocio*, un determinado servicio o, incluso, ser empleados como almacén de datos. Estos componentes se utilizan en la construcción de aplicaciones como si fueran piezas de un mecano y están diseñados para ser reutilizables.

Si bien con las tecnologías revisadas hasta el momento se pueden desarrollar básicamente los mismos servicios, a medida que estos servicios o aplicaciones Web adquieren relevancia, no están preparadas para hacerlo con un rendimiento adecuado o para poder aplicar técnicas de ingeniería del software que faciliten el desarrollo inicial y su posterior mantenimien-

to. Con la apuesta firme del mundo de los negocios por las tecnologías Web para desarrollar sus aplicaciones, han ido surgiendo y evolucionando diversas propuestas encaminadas a suplir estas necesidades operativas, como *servlets*, *java beans* o *COM*. Durante los siguientes apartados iremos analizando las opciones disponibles más extendidas.

3.3.1. Servlets

Los *servlets* son programas escritos en Java, se pueden invocar desde un cliente, de forma similar a como se invocan los programas CGI; se ejecutan en el seno del servidor Web, que actúa como un «contenedor de componentes»; realizan una determinada tarea; y generan una salida (página Web dinámica) que es recogida por el servidor Web y posteriormente reenviada al navegador que realizó la invocación de este componente.

Teniendo encuentra que un *servlet* es un objeto software que se invoca externamente, representa un modelo mucho más cercano al *modelo CGI* que al *modelo de páginas activas*, donde este objeto se encuentra incrustado dentro de código HTML estático. Sin embargo, la diferencia fundamental con respecto al modelo CGI clásico se encuentra en que un servlet se ejecuta dentro del contexto, y por lo tanto del control, del servidor o, ahora, contenedor Web (como por ejemplo: *Apache Tomcat*).

En el listado 3.5 se presenta el código fuente de un servlet, cuya interpretación por un navegador Web convencional puede verse en la figura 3.3.

Sus características y método de trabajo son los siguientes:

- **Portabilidad**. Al estar escritos en Java, los servlets garantizan la portabilidad entre diferentes plataformas que dispongan de la *Máquina Virtual Java*. También existe portabilidad entre los diferentes servidores Web, ya que el API Servlet define una interfaz estándar entre el servlet y el servidor Web que es independiente del fabricante. Por lo tanto, no hay especificaciones dependientes del servidor, como ocurre con la interfaz ISAPI/NSAPI o con las *páginas activas*.
- **Rendimiento**. Al igual que las extensiones del servidor ISAPI/NSAPI y a diferencia de CGI, los servlets son instanciados una única vez, pudiendo, a partir de entonces, atender diferentes peticiones. La instanciación del servlet se puede producir cuando tiene lugar la primera petición o estar ya instanciado desde la propia inicialización del servidor Web.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HolaMundoServlet extends HttpServlet {

    public void init(ServletConfig conf)
            throws ServletException {
        super.init(conf);
    }  // de initServlet

    public void service(HttpServletRequest req, \
            HttpServletResponse res)
            throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter salida = res.getWriter();

        salida.println("<html>");
        salida.println("<head>");
        salida.println("</head>");
        salida.println("<body>");
        salida.println("<h2>¡Hola a Todos!</h2;>");
        salida.println("</body>");
        salida.println("</html>");
    }  // de service
}  // de HolaMundoServlet
```

Listado 3.5. Contenido de la página activa `holaMundo.js` que muestra en el navegador un mensaje de bienvenida.

- **Concepto de sesión**. El servlet puede mantener información acerca de la sesión de usuario, variables o recursos —como por ejemplo: una conexión a la base de datos— entre diferentes conexiones dirigidas al mismo servlet e, incluso, entre varios de ellos.
- **Software distribuido**. Se pueden cargar indiferentemente y de forma transparente tanto desde un disco local como desde una dirección remota, respondiendo a la nueva filosofía de software distribuido. Los servlets pueden comunicarse entre sí y, por tanto, es posible una reasignación dinámica de la carga de proceso entre diversas máquinas, es decir, un servlet podría pasarle trabajo a otro servlet residente en otra máquina.

- **Ventajas inherentes a Java**. Además de la portabilidad, se hereda otras ventajas de este lenguaje idóneo para desarrollos en Internet, como es la orientación a objetos, la modularidad y la reutilización. Por otra parte, la responsabilidad de la gestión de memoria, tarea a cargo del programador en otros lenguajes, recae en el propio Java, facilitando la recolección automática de memoria no utilizada e impidiendo el acceso directo a la misma.
- **Multithread**. Dado que los servlets pueden manejar múltiples peticiones concurrentemente, es posible implementar aplicaciones cooperativas, como por ejemplo una aplicación de videoconferencia.

3.3.2. JavaBeans

Un *JavaBean,* o sencillamente un «bean», representa un modelo de componentes software especificado por Sun Microsystems dentro de su plataforma *Java 2 Platform, Standard Edition* (J2SE) para la construcción de aplicaciones Java. Estos componentes están pensados para poder ser reutilizados fácilmente en la creación de aplicaciones.

A diferencia de los servlets, los beans están diseñados para actuar de forma autónoma, ejecutándose dentro del contexto de un contenedor más genérico que el servidor Web. Se podría decir que esta propuesta marca el inicio de la imparable estrategia por desacoplar la lógica de negocio, no sólo del código HTML, sino también del propio servidor Web. Este nuevo contexto se denomina «Servidor de Aplicaciones». Sin embargo, dejaremos su desarrollo para el próximo apartado, en el que analizaremos propuestas mucho más específicas y maduras, inspiradas desde el comienzo para proporcionar soporte a este nuevo concepto.

Los JavaBeans, además de exponer sus propiedades para poder ser personalizados y utilizar los eventos para comunicarse con el sistema o con otros beans, también utilizan la serialización de objetos Java para soportar la persistencia —lo que les permite guardar su estado y restaurarlo posteriormente—. Además, sus métodos, que no son diferentes de otros métodos Java, pueden ser invocados desde otros beans o desde páginas JSP.

Los beans han sido diseñados para que todas las claves de su API puedan ser entendidas y gestionadas por herramientas de desarrollo visual, incluyendo el soporte para eventos, las propiedades y la persistencia. Son los llamados *beans visuales*, y están íntimamente relacionados con la interfaz gráfica de usuario.

En cualquier caso, en el contexto del servidor de aplicaciones hablamos de los *beans no-visuales*, que pueden ser utilizados de forma programática, conociendo su funcionalidad e interfaz. En este grupo se distinguen dos tipos por su funcionalidad: los que contienen una determinada lógica de negocio y los que sirven como almacén de datos a los que se accederá posteriormente, cuando se necesite la información que albergan, en otro punto de la aplicación Web.

Aunque los beans individuales pueden variar ampliamente en funcionalidad, desde los más simples a los más complejos comparten las siguientes características:

- **Introspección (introspection)**: permite que la herramienta de programación o IDE pueda analizar cómo trabaja el bean.
- **Personalización (customization)**: el programador puede alterar la apariencia y la conducta del bean y se permite cambiar los valores de las propiedades del bean para personalizarlo.
- **Eventos (events)**: informa al IDE de los sucesos que puede generar en respuesta a las acciones del usuario o del sistema, y también los sucesos que puede manejar.
- **Persistencia (persistente)**: un bean tiene que tener persistencia, es decir, implementar el interfaz *Serializable*.

Cuando el IDE carga un bean, usa el mecanismo denominado *reflection* para examinar todos los métodos, fijándose en aquellos que empiezan por **set** y **get**. El IDE añade las propiedades que encuentra a la hoja de propiedades para que el programador personalice el bean.

3.3.3. Objetos OLE, ActiveX y COM

Microsoft fue uno de los pioneros en poner en práctica las ventajas del concepto de objeto implementando el intercambio dinámico de datos (*Dynamic Data Exchange* —DDE) desde las primeras versiones de Windows. Como evolución de este concepto desarrolló un sistema de objeto distribuido junto con un protocolo que denominó (*Object Linking and Embedding* —OLE 1.0). Mientras que DDE se limitaba a transferir una cantidad concreta de información entre dos aplicaciones, OLE fue capaz de mantener enlaces activos entre dos documentos o incluso incrustar un tipo de documento dentro de otro.

El principal cometido de OLE era la gestión de documentos compuestos, pero también se podía emplear para transferir datos entre aplicaciones mediante la técnica de «arrastrar y soltar» (*drag and drop*) y operaciones del portapapeles (*clipboard*). Sin embargo, el concepto que popularizó esta tecnología en el entorno Web era el de incrustar (*embedding*) objetos multimedia (animaciones flash, archivos de vídeo, etc.) en páginas Web, dentro del código HTML.

El problema es que OLE 1.0 es una tecnología del lado del cliente, pues el objeto incrustado debía estar situado en el equipo en el que se iba a trabajar.

Posteriormente OLE 1.0 evolucionó a OLE 2.0 que, por motivos probablemente comerciales, pasó a denominarse ActiveX. Además, y para confundir más aun las cosas, OLE 2.0 se volvió a implementar sobre COM (*Component Object Model*).

COM representa la primera arquitectura software de componentes de Microsoft, diseñada para facilitar la reutilización de software y la interoperatividad de los componentes mediante la comunicación entre procesos y la creación dinámica de objetos, en cualquier lenguaje de programación que soporte dicha tecnología.

El término COM es a menudo usado en el mundo del desarrollo de software como un término que abarca tecnologías anteriores a las que había absorbido o de las que había evolucionado: OLE, *OLE Automation* o ActiveX; pero también podemos encontrar referencias a otras tecnologías que han evolucionado a partir de ella, como DCOM y COM+, que estudiaremos con más detalle en el capítulo 4.

3.4. CASO DE USO: TOMCAT

Tomcat es un contenedor de Servlets creado por el grupo Apache que implementa las especificaciones de *Java Server Pages* y *Java Servlet* definidas por SUN mediante el comité JCP *(Java Community Process)*. Un contenedor de Servlets es un marco de ejecución que maneja e invoca Servlets por cuenta del usuario. El servidor Tomcat también es conocido como servidor de Aplicaciones. Un JSP se compila antes de ser servido al usuario final y es transformado a un Servlet. Tomcat incluye el compilador *Jasper* que se encarga de esta tarea.

Podemos dividir los contenedores de Servlets en:

- *Contenedores de Servlets Stand-Alone (Independientes)*. Forman parte del servidor Web. Este es el modo por defecto usado por Tomcat.

- *Contenedores de Servlets dentro-de-Proceso*. El contenedor Servlet es una combinación de un plug-in para el servidor Web y una implementación de contenedor Java. El plug-in del servidor Web abre una máquina virtual de Java, JVM (*Java Virtual Machine*) dentro del espacio de direcciones del servidor Web y permite que el contenedor Java se ejecute en él. Si una cierta petición debiera ejecutar un Servlet, el plug-in toma el control sobre la petición y lo pasa al contenedor Java (usando JNI, *Java Native Interface*). Un contenedor de este tipo es adecuado para servidores *multi-thread* de un sólo proceso y proporciona un buen rendimiento pero está limitado en escalabilidad
- *Contenedores de Servlets fuera-de-proceso*. El contenedor Servlet es una combinación de un plug-in para el servidor Web y una implementación de contenedor Java que se ejecuta en una JVM fuera del servidor Web. El plug-in del servidor Web y la JVM del contenedor Java se comunican usando algún mecanismo IPC, *Inter Process Communication,* (normalmente sockets TCP/IP). Si una cierta petición debiera ejecutar un Servlet, el plug-in toma el control sobre la petición y lo pasa al contenedor Java (usando IPCs). El tiempo de respuesta en este tipo de contenedores no es tan bueno como el anterior, pero obtiene mejores rendimientos en otros aspectos (escalabilidad, estabilidad, etc.).

Tomcat puede funcionar por sí solo como servidor Web pero normalmente deja esta tarea a un servidor más especializado, como es el caso de Apache (actualmente se soportan los servidores Apache, IIS y Netscape). Esto significa que siempre que despleguemos Tomcat se deberá decidir cómo usarlo, y, si se seleccionan las opciones 2 o 3 descritas anteriormente, también necesitaremos instalar un adaptador de servidor Web.

3.4.1. Instalación

Para instalar el contenedor Web Tomcat es necesaria la instalación de la máquina virtual de java, JVM (www.java.sun.com). En general, las distribuciones de Linux suelen traer una JVM instalada. Se deberá consultar la compatibilidad con la versión de Tomcat. En caso de no ser compatible se debe instalar una JVM compatible.

Se puede descargar el archivo que contiene el paquete Tomcat comprimido directamente de `http://archive.apache.org/dist/tomcat/tomcat-5/v5.5.23/bin/jakarta-tomcat-5.5.23.tar.gz`.

En este caso, se va a utilizar la última versión de Tomcat 5, versión 5.5.23 (marzo 2007). En la actualidad, se está trabajando en las primeras versiones de Tomcat 6. Si el lector prefiriese usar dicha distribución, debe saber que se basa en Tomcat 5, pero que existen algunas modificaciones y mejoras (consultar la documentación oficial en `http://tomcat.apache.org/tomcat-6.0-doc/index.html`).

Para instalar Tomcat, simplemente se debe descomprimir el archivo en el directorio deseado, por ejemplo en el */usr*.

```
$ tar -xzfv jakarta-tomcat-5.5.23.tar.gz
```

Listado 3.6. Descompresión del servidor Tomcat.

Una vez descomprimido se creará un directorio llamado `jakarta-tomcat-5.5.23` con la estructura de Tomcat.

En este directorio se encontrarán los siguientes directorios:

- *Bin*. Contiene los *scripts* necesarios para arrancar y parar el servidor Tomcat.
- *Conf*. Contiene los archivos de configuración.
- *Logs*. Por defecto, es el directorio donde se guardan los archivos de log.
- *Webapps*. Directorio raíz donde se ubicarán las aplicaciones.

Antes de arrancar Tomcat se deben establecer ciertas variables de entorno para que Tomcat pueda ser ejecutado.

Como Tomcat funciona sobre la máquina virtual de Java se debe tener instalada la MVJ (`j2sdk 1.4` o superior de Sun) en nuestra máquina. Se puede obtener en `http://java.sun.com/j2se/` la distribución binaria necesaria para nuestro sistema. En este caso, se partirá de la premisa que ya se encuentra instalada la MVJ.

Ahora se debe declarar una variable de entorno denominada `JAVA_HOME`, la cual indica la ubicación del directorio donde se encuentra la máquina virtual de java (para la distribución de Knoppix utilizada en el libro, en el listado 3.7 se muestra como definir la variable `JAVA_HOME`).

```
$ export JAVA_HOME=/usr/lib/jvm/java-1.5.0-sun-1.5.0.10
```

Listado 3.7. Variable de entorno JAVA_HOME.

Posteriormente, se debe definir otra variable que indica la ubicación del Servidor Tomcat. Esta variable se llamará `CATALINA_HOME`

```
$ export CATALINA_HOME=/usr/apache-tomcat-5.5.23
```

Listado 3.8. Variable de entorno CATALINA_HOME.

Se debe tener en cuenta que si se exportan dichas variables sobre un terminal solamente tendrán valor local en esa sesión. Para no tener que definirlas cada vez que se abra una sesión o que se inicie el servidor, una forma de hacerlo es declararlas en el archivo `profile` del directorio */etc*.

Una vez configuradas las variables de entorno se puede arrancar el Servidor Tomcat. Para ello se utiliza el *script* `catalina.sh` ubicado en el directorio `bin`.

```
$ catalina.sh start
```

Listado 3.9. Comando para arrancar el servidor Apache Tomcat.

Una vez comprobado que el servicio ha sido lanzado correctamente, se puede abrir el navegador y acceder a la página inicial del Tomcat introduciendo la *URL* `http://localhost:8080`.

Para detener el servicio se utilizará el mismo *script* que para el arranque del servidor pero, con un parámetro distinto:

```
$ catalina.sh stop
```

Listado 3.10. Secuencia de parada del servidor Apache Tomcat.

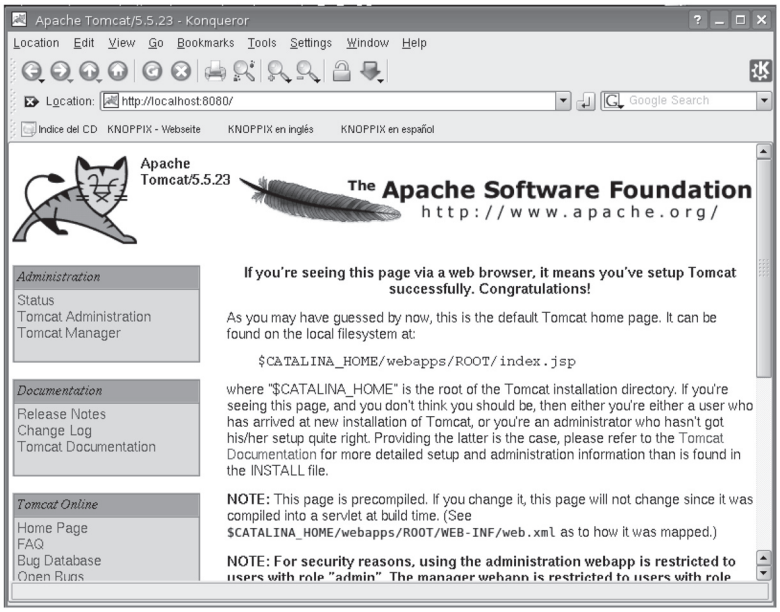


Figura 3.4. Página de presentación del servidor Tomcat.

3.4.2. Configuración básica

En el directorio *conf* se puede encontrar los archivos de configuración del Servidor Tomcat. El principal archivo de configuración es `server.xml`. Dicho archivo proporcionará la configuración inicial para los componentes Tomcat y especificará la estructura permitiendo que Tomcat se cree a si mismo con los componentes indicados.

El archivo `server.xml` contiene una serie de etiquetas con un significado muy concreto.

3.4.2.1. Etiqueta <server>

La etiqueta `<server>` representa el contenedor completo Tomcat. Es decir, el servidor Tomcat que se ejecuta en la maquina servidora. Los elementos que se encuentran por debajo especificarán parámetros de configuración para dicho contenedor. Además llevará una serie de atributos.

- El atributo `port` Indica el puerto por el cual se escuchan las peticiones de parada del servicio.
- El atributo `shutdown` es la cadena que debería recibir el servicio por el puerto indicado para detener el servicio.

3.4.2.2. Etiqueta <service>

La etiqueta `<service>` representa un conjunto de conectores que serán compartidos por un mismo contenedor Tomcat. Una etiqueta `<server>` puede tener una o más etiquetas `<service>`, los cuales tendrán un nombre único.

3.4.2.3. Etiqueta <connector>

La etiqueta `<connector>` representa las interfaces para comunicar un servicio con un cliente recibiendo las peticiones. Existen dos tipos diferentes de conectores: los que permiten a un navegador conectarse directamente al Tomcat (con el protocolo HTTP/1.1) y los conectores que lo hacen a través de un Web Server (mediante el protocolo AJP/1.3).

El primer conector se puede utilizar para hacer que Tomcat reciba las peticiones cuando se ejecute como servidor independiente, mientras que el segundo se utilizará cuando se use como *plug-in* de un Servidor Web.

Todos los conectores soportan los siguientes atributos comunes:

- El atributo `className` define la clase Java que lo implementa. Esta debería ser `org.apache.catalina.Connector` interface.
- El atributo `enableLookups` puede tener dos valores. *True* si se quiere buscar en el DNS para saber el nombre actual del cliente remoto. Se usará la opción *False* para obviar el DNS y devolver la IP en su lugar. Por defecto a *False*.
- El atributo `redirectPort` especifica el puerto al cual se redireccionarán las peticiones si el conector no soporta SSL. El primer conector (*Coyote HTTP/1.1*) puede tener otra serie de atributos:
- El atributo `acceptCount` indica el máximo número de peticiones encoladas cuando todos los procesos están ocupados. Cualquier petición recibida cuando la cola está completa será rechazada.
- El atributo `connectionTimeout` indica el número en milisegundos que la conexión puede esperar.
- El atributo `debug` indica el nivel de depuración.

- El atributo `maxProcessors` indica el número máximo de hilos que puede crear este conector (número máximo de peticiones simultaneas).
- El atributo `minProcessors` indica el número de hilos para recibir peticiones que serán creados cuando el conector sea lanzado por primera vez. Debería ser más pequeño que el atributo anterior.
- El atributo `port` indica el puerto TCP en el cual el conector se encontrará escuchando.

El conector `Coyote JK 2` permite comunicar el Servidor Web Apache con Tomcat redirigiendo las peticiones indicadas de Apache a Tomcat de una manera transparente.

La implementación estándar de `JK 2 Connector` está en la clase `org.apache.coyote.tomcat4.CoyoteConnector`, pero se debe especificar el atributo `protocolHandlerClassName`.

Los atributos específicos para dicha clase son:

- *acceptCount*. Máximo número de peticiones encoladas cuando todos los procesos están ocupados. Cualquier petición recibida cuando la cola está completa será rechazada.
- *debug*. Nivel de depuración.
- *maxProcessors*. Número máximo de hilos que puede crear este conector. Número máximo de peticiones simultaneas.
- *minProcessors*. Número de hilos para recibir peticiones que serán creados cuando el conector sea lanzado por primera vez. Debería ser más pequeño que el atributo anterior.
- *port*. El puerto TCP en el cual el conector se encontrará escuchando.
- *protocolHandlerClassName*. El valor de este atributo debería ser `org.apache.jk.server.JkCoyoteHandler` para usar el *JK 2 handler*.

3.4.2.4. Etiqueta <context>

La etiqueta `<context>` representa una aplicación Web, la cual corre en un *host virtual* particular. Cada *Web application* estará basada en un *Web application archive* (extensión *WAR*). Este tema está más relacionado con el desarrollo por lo que se puede consultar para más información en las siguientes URLs:

```
http://java.sun.com/products/servlet/download.html
http://tomcat.apache.org/tomcat-5.5-doc/developers.html
```

Se podrán definir tantos contextos como se necesiten dentro de un `HOST`. Cada contexto debería tener *context path* único definido por el atributo `path`. Se debería definir un contexto con el atributo `path` vacío, el cual sería el contexto raíz y el contexto por defecto, de tal forma que una petición sin contexto buscaría en éste. Los contextos también se pueden crear dentro de los archivos de configuración de las aplicaciones de manera automática. A partir de Tomcat 5, no se recomienda declarar los elementos `<context>` dentro del archivo `server.xml` puesto que cualquier modificación supone volver a lanzar el servidor para que vuelva a leer el archivo `server.xml`.

El elemento `context` se puede definir explícitamente de las siguientes maneras:

- En el archivo `$CATALINA_HOME/conf/context.xml`
- En el archivo `$CATALINA_HOME/conf/[enginename]/[hostname]/context.xml.default`. Se trata de una definición de la información de contexto de forma global que será cargada por todos los *webapps* de este *Host*.
- En un archivo *.xml* específico para cada aplicación en el directorio `$CATALINA_HOME/conf/[enginename]/[hostname]/`. El nombre del archivo, sin la extensión *xml*, será utilizado como la ruta de contexto de acceso a la aplicación. Por ejemplo, si se crea el archivo de contexto `ejemplo.xml`, entonces la *URL* de acceso a la aplicación será `http://localhost:8080/ejemplo`. El contexto de la aplicación raíz, por defecto `"/"`, suele denominarse `ROOT` (`ROOT.xml`).
- También se puede definir el contexto en un archivo `context.xml` dentro del directorio *META-INF* de cada aplicación individual.
- Por último, y también en desuso, consistiría en introducir el elemento `context` dentro de un elemento `<host>` en el archivo `server.xml`.

A continuación se definen los principales atributos para la configuración básica de un contexto:

- El atributo `className` indica el nombre de la clase Java que implementa la interfaz `org.apache.catalina.Context`. Si no se especifica, se usará la de por defecto.
- El atributo `cookies` tendrá un valor `true` si quieres que se usen cookies para identificar la sesión.

- El atributo `crossContext` tendrá un valor `true` para poder interactuar con otras aplicaciones Web que corran en este *host virtual*.
- El atributo `docBase` indica el directorio para esta aplicación Web. También conocido como `DocumentRoot`. Se puede especificar un *path* absoluto para este directorio o un *path* relativo al directorio base del propio *Host* (atributo `appBase` de `Host`).
- El atributo `path` indica el *path* que se debe incluir en la *URL* para acceder a la aplicación. Si se especifica el valor `""`, se estará definiendo la aplicación Web por defecto para dicho *Host*.
- El atributo `reloadable` tendrá un valor `true` para indicar que si se realiza algún cambio en `/WEB-INF/classes/` y `/WEB-INF/lib` (directorios de desarrollo de aplicaciones) automáticamente actualizará esos cambios.
- El atributo `debug` indica el nivel de *log* para el contexto.

```
<Context path="/prueba"
docBase="${Catalina.home}/server/webapps/prueba" privileged="true"
reloadable="true" antiResourceLocking="false"
antiJARLocking="false"/>
```

Listado 3.11. Ejemplo de contexto (`ejemplo.xml`).

La aplicación estaría en el directorio `webapps/ejemplo` y se haría referencia a ella con `http://localhost:8080/ejemplo/pagina.jsp`.

3.4.2.5. Etiqueta <host>

La etiqueta <`host`> representa un *host virtual* el cual asocia un nombre de red con un servidor en particular donde está corriendo Tomcat. El nombre debería estar registrado en un sistema de resolución de nombres (Normalmente DNS).

Los atributos básicos son:

- El atributo `appBase` indica el directorio de aplicaciones base para este *host virtual*. Éste es el nombre del *path* que contiene las aplicaciones Web para ser ejecutadas en el *host virtual*. Se puede especificar un *path* absoluto o uno relativo al directorio `$CATALINA_HOME`.

- El atributo `autoDeploy` indica si las aplicaciones Web deberían ser cargadas automáticamente por el configurador del *host* (`true`)
- El atributo `name` indica el nombre de red para el *host virtual*, tal como esta registrado en el servidor de resolución de nombres. Uno de los *Host* incluidos en un `engine` deberá tener el mismo nombre que el atributo `defaultHost` colocado en el *Engine*.

```
.....
<Host  name="localhost"  debug="0"  appBase="webapps"
unpackWars="true" autodeploy="true">
   <Logger className="org.apache.catalina.logger.FileLogger"
   directory="logs" prefix="localhost_log." suffix=".txt"
   timestap="true"/>
   <Context>.......</Context>
   ........................
   <Context>.......</Context>
</Host>
.....
```

Listado 3.12. Ejemplo de la etiqueta host.

El `logger` indicará que se creará un archivo de *log* en el directorio *logs* llamado `localhost_log.txt` que controlará el *log* a nivel de este *host*.

Un `logger` se puede declarar de la misma manera a nivel de contexto indicando el *log* para ese contexto en particular.

3.4.2.6. Etiqueta <engine>

La etiqueta `<engine>` representa el mecanismo completo de procesamiento de peticiones asociado con un servicio Tomcat en particular. Este recibe todas las peticiones desde uno o varios conectores y devuelve la respuesta al conector par que llegue al usuario final.

- El atributo `defaultHost` indica el nombre del *host* por defecto, el cual identifica el *host* que procesará las peticiones dirigidas a los nombres de *host* en dicho servidor. Estos nombres no son configurados en este archivo.
- El atributo `default` indica el nivel de *log*.

- El atributo `name` indica el nombre lógico del `engine`. Usado en el *log* y mensajes de error.

```
.....
<Engine name="standalone" defautlHost="localhost" debug="0">
</Engine>
.....
```

Listado 3.13. Ejemplo de *engine*.

El proceso que implementa el *engine* analizará la cabecera y se lo pasará al *host virtual* apropiado.

3.4.3. La consola de administración de Tomcat

Tomcat, hasta la versión 5, tiene una consola de administración que permite configurar y mantener el servicio Tomcat, bases de datos, usuarios registrados, contextos.

El acceso a esta consola de administración se realiza mediante un enlace en la página principal del servicio cuando Tomcat actúa como servidor Web.

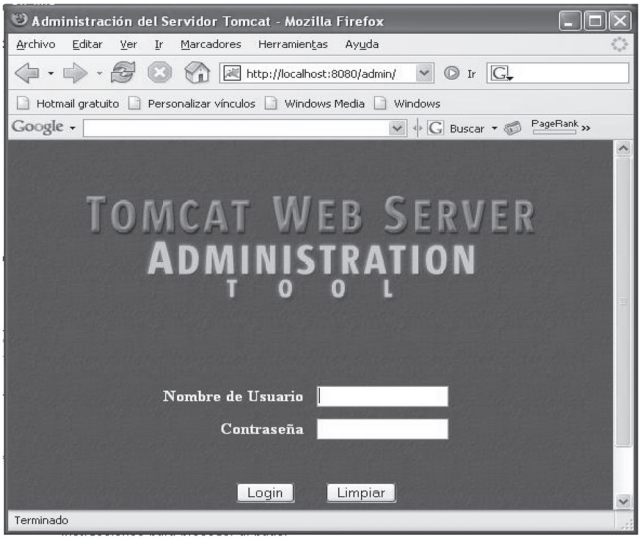


Figura 3.5. Consola de configuración del servidor Tomcat.

El acceso a la consola se realiza utilizando un usuario y una contraseña. La configuración de estos usuarios se realiza en el fichero `tomcat-users.xml` y tiene la siguiente estructura:

```
<tomcat-users>
  <role rolename="tomcat" />
  <role rolename="role1" />
  <role rolename="manager" />
  <role rolename="admin" />
  <user username="tomcat" password="tomcat" roles="tomcat" />
  <user username="role1" password="tomcat" roles="role1" />
  <user username="both" password="tomcat" roles="tomcat,role1" />
  <user username="admin" password="" roles="admin,manager" />
</tomcat-users>
```

Listado 3.14. Listado del archivo tomcat-user.xml.

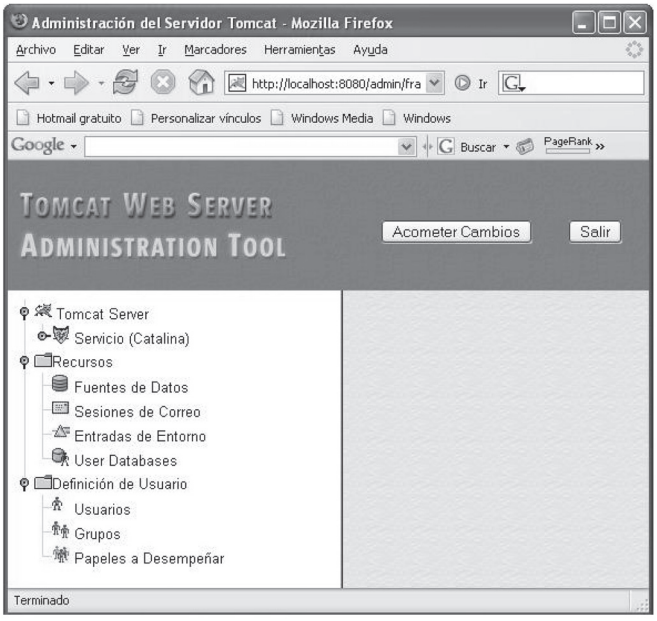


Figura 3.6. Consola de configuración del servidor Tomcat.

Se debe configurar los roles apropiados para los usuarios y a los usuarios con su password que tienen acceso al sistema.

Una vez se accede a la consola se podrá configurar de forma más amigable todas las características de Tomcat vistas en el punto anterior.

3.4.4. Despliegue de aplicaciones

Aunque no es el objetivo de este libro a continuación se describe brevemente como desplegar una aplicación Web en el servidor Apache Tomcat.

Una aplicación Web Java se compone de una estructura estándar jerárquica de directorios y archivos. Dicha estructura puede encontrarse desempaquetada o empaquetada como una Web ARchive (extensión *.war*). El primero es usado durante la etapa de desarrollo mientras que el segundo es usado para distribuir la aplicación en entornos de producción. Para saber más sobre como estructurar una aplicación Web acceder a `http://tomcat.apache.org/tomcat-4.0-doc/appdev/deployment.html`.

Una vez se tiene la estructura de la aplicación, en su formato empaquetado o no, se debería desplegar en el contenedor Tomcat. Algunas de las maneras en las que una aplicación Web puede ser desplegada en Tomcat son:

- Copiando la estructura en su formato desempaquetado en un subdirectorio del directorio `$CATALINA_HOME/webapps/`. El servidor Apache Tomcat asignará una ruta de contexto para la aplicación (`context path`) basada en el nombre del subdirectorio elegido. Se debe reiniciar Tomcat cuando se despliegue o se actualice la aplicación si no se ha configurado para el despliegue en caliente (*hot-deploy*).
- Usando la consola de gestión de aplicaciones *Tomcat Manager*. Esta consola permite desplegar y eliminar aplicaciones sin necesidad de tener que reiniciar el servidor Tomcat. Para ello se indica la ruta relativa que se debe introducir en la *URL* para acceder a la aplicación, la ruta donde se encuentra la aplicación y la *URL* del archivo de configuración XML si es necesaria.

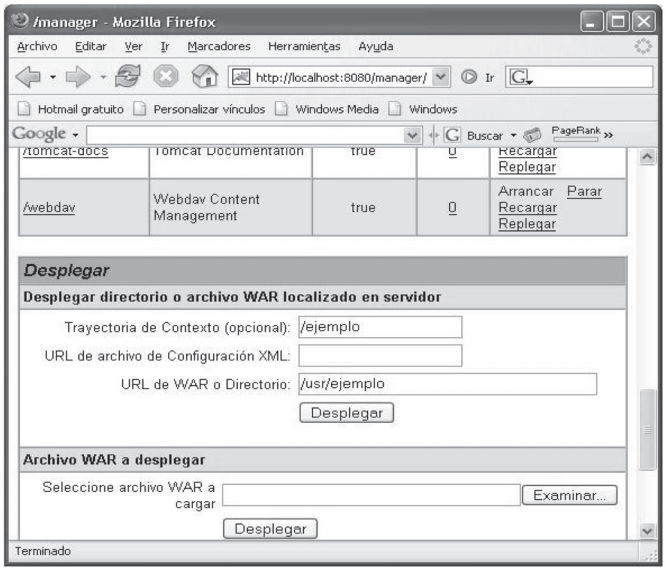


Figura 3.7. Consola de gestión del servidor Tomcat.

3.4.5. Integración de Apache y Tomcat

En apartados anteriores se ha descrito la manera de configurar tanto Apache 2.0 como Tomcat para que funcionen de forma independiente. Apache como servidor de páginas Web estáticas y Tomcat como servidor independiente de aplicaciones Web JSP/Servlets. En este apartado se describe como unir ambas aplicaciones para extraer el mayor potencial de cada una de ellas. Apache como servidor Web especializado contiene una gran cantidad de funcionalidades y extras que no posee Tomcat. Tomcat como contenedor Web es capaz de servir páginas dinámicas que ejecutan código en función del contexto de la aplicación Web. En este caso de uso se utilizará Apache 2 como servidor Web principal, y éste hará uso de Tomcat 5 como *plug-in* para servir páginas dinámicas JSPs.

Tomcat admite dos modos de funcionamiento:

- *Contenedor independiente*: actúa como un servidor Web autónomo. El problema es que Tomcat es menos rápido, robusto y configurable que Apache, por lo que este modo suele reservarse para desarrollo y depuración.

- *Módulo para un servidor Web existente* (Apache en el caso que se describe): en este caso, el servidor Web se encargaría de servir todo el contenido, acudiendo a Tomcat solamente para interpretar *Servlets/JSP*.

En general, el funcionamiento de un Servidor Web, como Apache, se reduce a la espera de peticiones de un cliente HTTP y, cuando éstas llegan, realizar las operaciones necesarias para proporcionar el contenido solicitado. La cooperación con un contenedor de Servlets añade algún cambio en este proceso, que sería ahora el siguiente:

- Apache debe cargar la librería del conector de Tomcat e inicializarlo (antes de servir peticiones).
- Cuando llega una petición, necesita comprobar si se trata o no de una solicitud correspondiente a un Servlet/JSP (usualmente basándose en algún patrón de la *URL*). En caso de serlo, deberá pasar el control al conector.
- Dicho conector deberá estar correctamente configurado para conocer dónde dirigir esas peticiones y reenviárselas a Tomcat.
- Las solicitudes de gestión de Servlets provenientes de Apache son recibidas por instancias de objetos denominados `workers`, que son generadas por Tomcat.

Según lo anteriormente comentado, un conector o adaptador es el encargado de gestionar la comunicación entre el Servidor Web y un módulo con el que éste interopera, como por ejemplo Tomcat. Este adaptador se compila, dando lugar a una librería dinámica que será cargada con Apache.

El proyecto JK es un subproyecto del grupo Jakarta-Tomcat, que permite crear conectores entre Tomcat y distintos Servidores Web. Los distintos conectores desarrollados por este proyecto son los siguientes:

- El módulo `mod_jk` (*conector JK*) se usa con Apache 1.3 y 2.0. El conector JK se comunica con Apache mediante el protocolo JK, también conocido como protocolo AJP (*Apache JServ Protocol*). Para la versión 2.0 de Apache se ha desarrollado una versión modificada de ese conector, denominada *Coyote JK2* consistente en una re-estructuración de JK para permitir una simplificación de la configuración.
- El modulo `isapi` redirector para IIS (Internet Information Server).

- El modulo `nsapi` redirector para Netscape/iPlanet.
- El modulo `dsapi` redirector para Domino.
- De acuerdo con lo anterior, se debe obtener la librería `mod_jk`, compatible con la versión "Apache 2.0" instalada en el sistema. La librería `mod_jk` correspondiente al adaptador Web para Tomcat en algunos casos puede encontrarse ya compilada para ciertos entornos concretos o si no debe obtenerse su código fuente, listo para ser compilado. Tanto las librerías compiladas como el código fuente se pueden obtener de `http://archive.apache.org/dist/jakarta/tomcat-connectors/` y la documentación del conector *JK 2* desde `http://tomcat.apache.org/connectors-doc/`. Es recomendable obtener la versión binaria si existe para la distribución que se esté utilizando. Desde la *URL* `http://tomcat.apache.org/download-connectors.cgi` se puede descargar el módulo binario para varias distribuciones. En concreto, en este caso, se está utilizando el módulo binario `http://apache.rediris.es/tomcat/tomcat-connectors/jk/binaries/linux/jk-1.2.21/mod_jk-1.2.21-apache-2.2.x-linux-i686.so`

3.4.5.1. Configuración de Tomcat

Esta parte es bastante sencilla. Se debe configurar el archivo `server.xml` para que esté activo el conector `Coyote JK 2`. En la configuración por defecto viene este conector activo o comentado por lo tanto no habría que tocar nada o como mucho descomentarlo (siempre que se vaya a utilizar dicha configuración) En el caso que se crease una configuración personalizada, simplemente, se debería copiar esa parte debajo del service correspondiente.

```
<!-- Define a AJP 1.3 Connector on port 8009 -->
<Connector  port="8009"  enableLookups="true"  redirec-
tPort="8443" protocol="AJP/1.3"/>
```

Listado 3.15. Declaración del conector AJP1.3.

Como se observa, se indica a Tomcat que las peticiones para este conector vendrán a través del puerto 8009.

3.4.5.2. Instalación de mod_jk

Este módulo debe ser cargado por Apache 2.0. Tal como se describió en el capítulo de Apache 2.0, éste realiza la carga de módulos utilizando archivos `.load` y `.conf` en un directorio particular de Apache (en el caso de la distribución usada, Knoppix, en */etc/apache2/mods-enabled*). El archivo `.load` se encarga de la carga del módulo correspondiente y el `.conf` de la configuración necesaria para que el módulo funcione correctamente. Primero se ha copiado el módulo binario descargado al directorio de módulos de apache, aunque podría ubicarse en cualquier directorio.

```
# cp mod_jk-1.2.21-apache-2.2.x-linux-i686.so
/usr/lib/apache2/modules/mod_jk2.so
```

Listado 3.16. Ubicación del módulo `mod_jk2.so`.

Posteriormente se crean los archivos necesarios de carga del módulo (listado 3.17) y de configuración (listado 3.18). En el primero, simplemente, se reindica la carga del módulo y su ubicación.

```
# Carga del modulo correspondiente
LoadModule jk_module /usr/lib/apache2/modules/mod_jk2.so
```

Listado 3.17. Listado del archivo mod_jk.load.

En el archivo de configuración se le indican los parámetros de configuración del módulo (ubicación del archivo `workers.properties`, archivo de *log*, etc.). Se puede obtener más información en la *URL* `http://tomcat.apache.org/connectors-doc/webserver_howto/apache.html`.

```
# Mod_jk settings
# Información de las directivas en:
# http://jakarta.apache.org/tomcat/connectors-doc/config/apache.html
#
#Ubicación del archivo workers.properties,
JkWorkersFile "/etc/apache2/workers.properties"
#Configuración de Logs
JkLogFile "/etc/apache2/mod_jk.log"
JkLogLevel debug
#Genera automáticamente las directivas Alias de Apache para el
#contenedor de aplicaciones de Tomcat
JkAutoAlias /usr/java/tomcat/webapps
#Redirección de todos los jsp a al worker nombrado como apache
en el #workers.properties
JkMount /*.jsp apache
#End of mod_jk settings
```

Listado 3.18. Listado del archivo mod_jk.conf.

3.4.5.3. Configuración de Workers

Un Tomcat workers es una instancia de un servidor Tomcat que está esperando que le lleguen peticiones para ejecutar Servlets provenientes de un Servidor Web.

El archivo `workers.properties` proporciona la información necesaria al módulo `mod_jk` sobre la ubicación de Tomcat en la red y el puerto en el que realizar la escucha. Su nombre y ubicación puede ser cualquiera siempre que se informe al módulo `mod_jk` convenientemente.

```
# Información de las directivas en:
# http://jakarta.apache.org/tomcat/connectors-
#doc/config/workers.html
worker.tomcat_home=$CATALINA_HOME
worker.java_home=$JAVA_HOME
ps=/
worker.list=apache
worker.apache.port=8009
worker.apache.host=localhost
worker.apache.type=ajp13
```

Listado 3.19. Listado del archivo `worker.properties`.

En el listado 3.19 se crea un *worker* llamado `apache` (se podrían crear los workers necesarios en función de las necesidades separándolos por comas y un espacio `worker.list=apache, balanceo`). Posteriormente se definen los puntos de acceso de cada worker (más información sobre *workers* en `http://tomcat.apache.org/connectors-doc/generic_howto/workers.html`).

Una vez realizado el workers se pueden realizar las pruebas con los ejemplos de JSP que contiene TOMCAT para verificar el correcto funcionamiento accediendo a la *URL* `http://localhost/index.jsp`. Si muestra la página principal de Tomcat se ha configurado de forma correcta el módulo de conexión.

4. SERVIDOR DE APLICACIONES

Hasta el momento, la mayor parte de las tecnologías que se han ido introduciendo no han hecho otra cosa que engordar y engordar el servidor Web, cargándolo con todo el trabajo adicional a base de parches o añadidos de diversa naturaleza —justo en contra de la idea original, donde un protocolo muy básico y una sencilla aplicación servidora gestionaban la presentación de contenidos a, potencialmente, cualquier equipo que se pudiera conectar a la red y dispusiera de un sencillo navegador Web.

A medida que las primeras y sencillas páginas Web se han transformado en sofisticadas «aplicaciones Web» que proporcionan soporte a organizaciones de cualquier tamaño, parece una mejor opción devolver este escenario a su estado inicial y buscar soluciones más ajustadas a los nuevos requerimientos. Según esto, para gestionar todas las posibles tareas de índole dinámica que un servidor Web deba realizar, se propone la colocación, *detrás de él,* un nuevo sistema especialmente concebido para resolverlas con solvencia y liberar al servidor Web de toda carga extra. Es lo que se conoce como «servidor de aplicaciones».

Este enfoque no implica que no se vayan a poder seguir empleando todas las tecnologías anteriormente estudiadas, sino que la tendencia será ir migrando las aplicaciones hacia la utilización del servidor de aplicaciones a medida que vayan creciendo los requerimientos de escalabilidad, eficiencia y adaptabilidad, por mencionar algunos de ellos, de las nuevas aplicaciones Web.

La filosofía general de funcionamiento del nuevo tándem servidor Web más servidor de aplicaciones es la siguiente: las peticiones realizadas al servidor Web que deban generar contenido estático serán despachadas por el propio servidor Web, como de costumbre, pero las peticiones que deban generar contenido dinámico serán delegadas en el servidor de aplicaciones; éste interactuará con los recursos que se precisen, ejecutará la lógica de negocio asociada y le pasará la respuesta en formato HTML o XML al servidor Web que, a su vez, la remitirá al cliente que invocó la petición (ver figura 4.1).

Sin embargo, todavía podemos llegar un poco más lejos, de hecho, hasta la situación actual, donde nos encontramos con el que ha resultado ser el escenario más adecuado para desplegar complejas aplicaciones sobre Internet: estamos hablando de los «sistemas distribuidos».

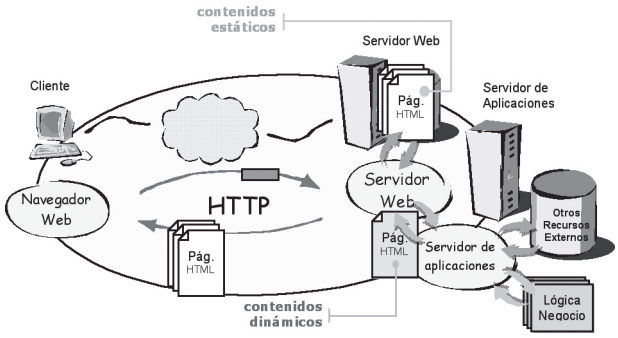


Figura 4.1. Relación entre un *servidor Web* y un *servidor de aplicaciones*.

En este tipo de escenarios, las aplicaciones se desarrollan como un conjunto de componentes software que encapsulan la lógica de negocio y que se ejecutan distribuidos sobre un conjunto de servidores de aplicaciones interconectados a través de una red de comunicaciones basada, generalmente, en TCP/IP (una intranet o, incluso, la propia Internet).

En un entorno de estas características seguimos interesados en perfeccionar el modelo de componentes pero, sobre todo, estamos más que interesados en que la complejidad de las infraestructuras que tienen que servir de soporte quede oculta para las aplicaciones y para los desarrolladores, y que su gestión no se convierta en un verdadero calvario para los administradores de sistemas.

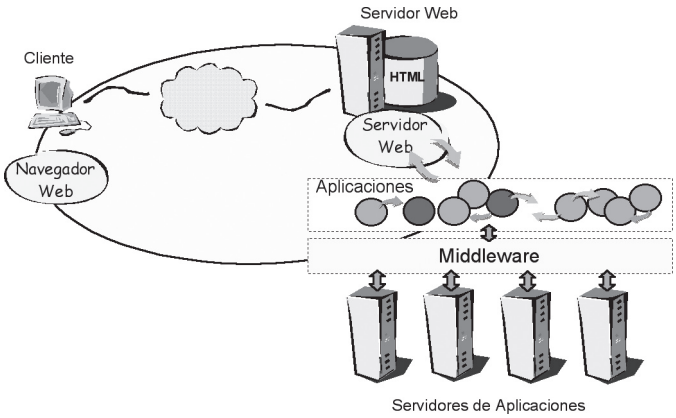


Figura 4.2. Capa de *middleware* en el contexto de un sistema distribuido

De igual forma que en un equipo aislado es el sistema operativo el responsable de ocultar a las aplicaciones los detalles concretos de los dispositivos y componentes físicos, el conjunto de tecnologías y servicios que proporcionan la transparencia deseada (ver figura 4.2), ya no sólo sobre un único equipo, sino sobre todo un conjunto de equipos, así como de la propia red de comunicaciones e, incluso, de los sistemas operativos de cada una de ellos, lo denominamos «middleware».

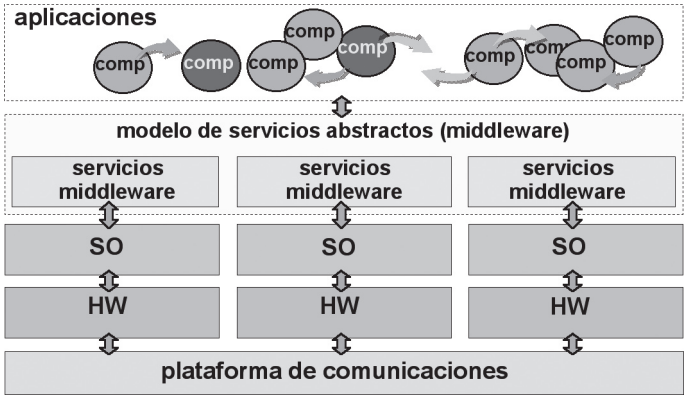


Figura 4.3. Modelo general de capas del sistema con middleware.

4.1. MIDDLEWARE

El middleware se define como el nivel lógico del sistema que proporciona una abstracción sobre la infraestructura que le da soporte a las aplicaciones, dotando de transparencia de ubicación e independencia de los detalles del hardware de los computadores, del sistema operativo, de la red de interconexión y de los protocolos de comunicaciones (figura 4.3) e, incluso, del lenguaje de programación utilizado para su desarrollo. Desde el punto de vista del programador de aplicaciones, el middleware establece un modelo de programación sobre bloques básicos arquitectónicos, utilizando protocolos basados en mensajes para proporcionar abstracciones de nivel superior.

En cualquier caso, el middleware se puede estudiar desde dos puntos de vista bien diferenciados: desde el punto de vista de la infraestructura y servicios que lo conforman o desde el punto de vista del desarrollador de aplicaciones. Veamos a continuación cada uno de ellos.

4.1.1. Infraestructura de servicios

Desde el punto de vista de la propia infraestructura que conforma el *middleware*, éste viene definido por los elementos que componen el núcleo de servicios, ubicado entre el sistema operativo local y las aplicaciones, y en la que resaltan los siguientes elementos (figura 4.4): el modelo de representación de datos, el modelo de comunicaciones —en el que se incluye el protocolo de comunicaciones y el modelo de invocación de métodos remotos— y los servicios fundamentales que proporciona la plataforma.

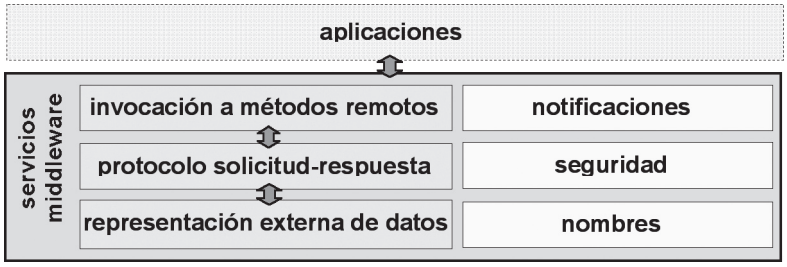


Figura 4.4. Alguno de los principales componentes que forman la capa de servicios middleware que proporciona soporte y transparencia a las aplicaciones.

4.1.1.1. Modelo de representación de datos

El modelo de comunicación proporciona los mecanismos para que dos aplicaciones potencialmente ubicadas en equipos diferentes puedan interactuar entre sí de forma transparente. Al mismo tiempo, estos mecanismos determinan cómo se deberá establecer la comunicación y cómo tendrán que prepararse cliente y servidor para poder realizarla.

Independientemente de la forma de comunicación utilizada, las estructuras de datos y los atributos de los objetos deben ser convertidos en una secuencia de *bytes* antes de su transmisión y reconstruidos posteriormente en el destino. Para hacer posible que dos computadores puedan intercambiar información, deben acordar previamente un esquema común o incluir la especificación del formato de emisión en el propio mensaje.

Al estándar acordado para la representación de los valores y estructuras de datos a transmitir en los mensajes se denomina *representación externa de datos* —por ejemplo, CDR de CORBA.

4.1.1.2. Modelo de comunicación

El protocolo petición-respuesta define el nivel adecuado para establecer una comunicación típica según la arquitectura cliente/servidor en sistemas distribuidos basados en UDP o TCP. El propio mensaje de respuesta se considera como un reconocimiento del mensaje de petición, evitando de este modo las sobrecargas de los mensajes de reconocimiento adicionales.

Sobre esta capa de protocolo se construyen los modelos de comunicación de llamada a procedimiento remoto (RPC) y el modelo de invocación a un método remoto (RMI). RPC permite a programas cliente invocar procedimientos pertenecientes a programas servidor que generalmente ejecutan en computadores distintos. Este modelo evoluciona, posteriormente, permitiendo que objetos de diferentes procesos se comuniquen con los métodos de otros objetos.

En la figura 4.5 se puede observar una arquitectura de comunicaciones más o menos genérica de un middleware. Cuando se crea un componente que debe ser accedido de forma remota, debe crearse también un objeto *proxy* de la interfaz de dicho componente. De esta forma, el middleware puede presentar a la aplicación cliente una interfaz uniforme, con independencia de dónde se encuentre realmente el componente al que desea acceder, encargándose de resolver todos los problemas de

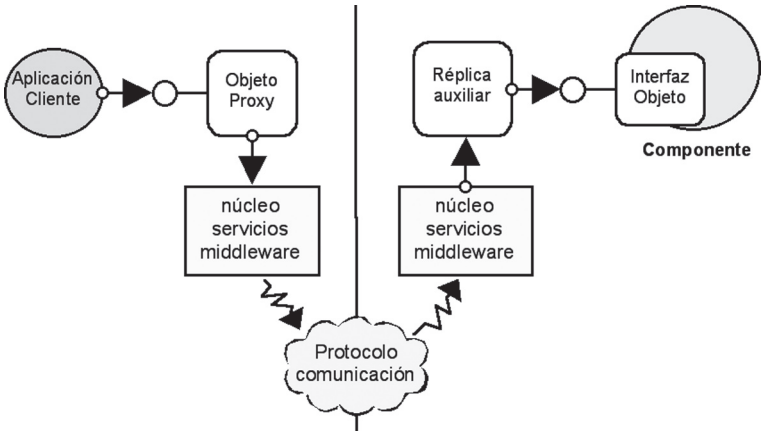


Figura 4.5. Arquitectura de comunicaciones middleware.

redirección de las solicitudes y respuestas, de localización de los componentes o de que los componentes se encuentran cargados en memoria y listos para responder.

4.1.1.3. Servicios comunes

Para las aplicaciones que se ejecutan apoyándose en el middleware, éste representa fundamentalmente un conjunto de servicios a los que pueden acceder o a partir de los cuales pueden acceder a otros servicios y recursos. El mecanismo de comunicación estudiado en el punto anterior es uno de estos servicios, quizá uno de los más importantes, pero no el único. De hecho, cada fabricante propondrá un conjunto de servicios que estime convenientes y que no tienen por qué coincidir de un middleware a otro. Por suerte, existe una serie de servicios que se pueden encontrar de una forma u otra en las diferentes propuestas de middleware para componentes software distribuidos. A continuación se realiza una descripción breve de los más comunes: servicios de nombres, de eventos y notificaciones y de seguridad.

Servicio de nombres

Su objetivo es almacenar los atributos de los objetos en un sistema distribuido —nombre y dirección entre otros—, devolviendo esos atributos

cuando se realiza una búsqueda sobre cierto objeto. Los principales requisitos que debe poseer un servicio de nombres son: habilidad para manejar un número arbitrario de nombres, persistencia, alta disponibilidad y tolerancia a fallos. Ejemplos de este tipo de servicio son: X.500, Jini, DNS, LDAP, etc.

Servicio de eventos y notificaciones

Este servicio extiende el modelo local de eventos al permitir que varios objetos en diferentes ubicaciones puedan ser notificados de los eventos que tienen lugar en un objeto. Emplean el paradigma *del tablón de anuncios,* en el que un objeto que genera eventos publica el tipo de eventos que ofrece para su observación. Los objetos que desean recibir notificaciones de otro objeto se suscriben a los tipos de eventos que les interesan.

Servicio de seguridad

Los mecanismos de seguridad se basan esencialmente en la criptografía de clave pública y de clave secreta. Los recursos se protegen mediante mecanismos de control de acceso. Se pueden mantener los derechos de acceso en listas de control (ACL) asociadas a conjuntos de objetos. Ejemplos de este servicio son: el protocolo de autenticación Needham-Schroeder, Kerberos, SSL y Millicent.

4.1.2. Modelo de programación y de componentes

Desde el punto de vista del diseñador de aplicaciones, el middleware representa un modelo de programación al que le incumbe definir nítidamente la estructura que deben tener los componentes software a diseñar, así como los mecanismos que deben emplearse para acceder a los recursos y a los servicios que proporciona la plataforma.

Los componentes software se pueden considerar piezas de software diseñadas para ser reutilizadas en diferentes planos: reutilización de código, de funcionalidad, de tipos. Los componentes software distribuidos, además, hacen especial hincapié en su ejecución transparente sobre entornos distribuidos. Están creados para encapsular una determinada lógica de negocio, un determinado servicio (como puede ser acceso a una determinada arquitectura) o, incluso, ser empleados como almacén de datos. Estos componentes se utilizan en la construcción de aplicaciones como si fueran piezas de un mecano.

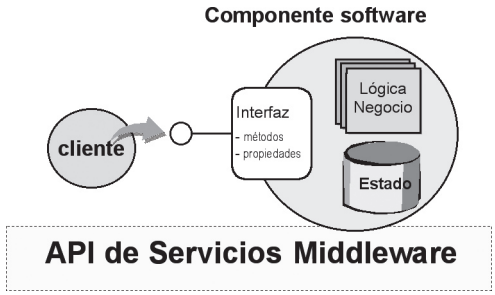


Figura 4.6. Estructura básica de un componente software y su relación con la plataforma.

Todas estas características resultan fundamentales para diseñar, desarrollar y mantener las actuales aplicaciones Web, sobre todo en el ámbito empresarial. Gracias a ellas se pueden aplicar técnicas de ingeniería del software sofisticadas y se facilita enormemente la adaptación de la aplicación a la situación cambiante del entorno. Una aplicación distribuida basada en el modelo de middleware permite una asignación de recursos muy ajustada a medida que éstos vayan necesitándose y, siempre que los cambios puedan localizarse en unos componenetes software determinados, su modificación tiene un impacto relativamente escaso sobre el global de la aplicación.

Un componente software debe poseer una estructura bien definida (ver fig. 4.6): una *interfaz* en la que se definen los métodos y propiedades que soporta el componente y a través de la cual los clientes pueden acceder; una *lógica de negocio* o lógica de aplicación que recoge la funcionalidad o el comportamiento del componente, tanto la que muestra al exterior a través de su interfaz, como la que presenta internamente; y un *estado* que le permite almacenar información que permite que dos instancias de un mismo componente puedan adquirir su propia entidad.

Por otra parte, el modelo de programación también determina cómo accederán estos componentes software a los servicios que ofrece el middleware y que, en este caso, consiste en la provisión de una *interfaz de programación (API) de servicios*.

4.1.2.1. Estructura de un componente

Un sistema orientado a objetos consta de un conjunto de componentes que interaccionan entre sí, cada uno de los cuales consiste en un conjunto de propiedades y un conjunto de métodos. Un objeto se comunica con otro

objeto invocando sus métodos, generalmente pasándole argumentos y recibiendo resultados. Así, cada componente se puede estructurar en tres elementos: una interfaz, una lógica de aplicación o de negocio y un estado:

Interfaz

La interfaz de un componente proporciona una definición de las signaturas de sus métodos —los tipos de sus argumentos, valores devueltos, excepciones, etc.— sin especificar su implementación. Establece, asimismo, qué métodos y propiedades están accesibles para el resto de componentes y objetos. Por supuesto, nada impide que un mismo objeto implemente distintas interfaces a la vez.

Cada componente se implementa de forma que oculta todo su estado y funcionalidad, excepto aquél que se hace visible a través de su interfaz en términos de *propiedades* y *métodos*, respectivamente. Esto permite modificaciones en la implementación de la lógica del componente o de su estructura interna —siempre que su interfaz se mantenga inalterada— con un impacto mínimo sobre el resto de la aplicación de la que forma parte.

Para que distintos componentes —implementados por distintos equipos de desarrollo y, posiblemente, con diferentes lenguajes de programación— puedan interactuar entre sí es necesario que sus interfaces estén definidas de forma homogénea. Para facilitar esto, el modelo de programación introduce los lenguajes de definición de interfaces (IDL) que proporcionan una notación específica en la que, por ejemplo, los parámetros de un método se describen como de entrada o salida y utilizando su propia especificación de tipos.

Estado

El estado de un componente consta de los valores de sus variables de instancia. En el paradigma de la programación orientada a objetos, el estado de un programa se encuentra fraccionado en partes separadas, cada una de las cuales está asociada con un objeto. El estado de un objeto está accesible sólo para los métodos del objeto, es decir, que no es posible que métodos no autorizados actúen sobre su estado.

Algunas implementaciones de middleware, como CORBA, permiten empaquetar y almacenar los objetos junto con su estado en un momento dado —en los llamados almacenes de objetos persistentes—, de forma que métodos de otros objetos puedan activarlos por invocación a través de su interfaz.

En general, se almacenan en cualquier momento en el que se encuentren en un estado consistente, con lo que dotan al sistema de tolerancia a fallos.

Lógica de negocio o comportamiento

El comportamiento define la funcionalidad del componente a través de una serie de métodos que recogen la lógica de aplicación o de negocio que encapsula y a la cual se puede acceder, en caso de estar disponible para otros componentes, a través de las interfaces definidas.

4.1.2.2. Acceso a los servicios

Una plataforma middleware debe proporcionar un mecanismo que permita a los componentes software acceder a los servicios que proporciona. El método más habitual, sobre todo con el objetivo de dotar a los desarrolladores de una independencia adecuada con respecto a los lenguajes y herramientas de programación que utilicen, es a través de interfaces de programación de aplicaciones (API).

4.2. PLATAFORMAS ACTUALES

Aunque desde el punto de vista más amplio podríamos definir un *servidor de aplicaciones* como «uno o más servidores de red, dedicados a ejecutar ciertas aplicaciones software, de forma que sus componentes pueden comunicarse entre sí de forma transparente», el término también hace referencia al propio software instalado en estos servidores cuyo objetivo es servir de soporte para la ejecución de las aplicaciones mencionadas. Es por esta función de soporte por la que este software de apoyo se denomina también *plataforma middleware*.

En la práctica, una de las plataformas que más ha contribuido a esta visión es J2EE de *Sun Microsystems*, razón por la cual en muchos entornos se considera un sinónimo de *servidor de aplicaciones*. Sin embargo, puesto que J2EE en realidad es un conjunto de especificaciones, en la actualidad podemos encontrar múltiples alternativas de implementación de otros fabricantes y, lo que es más interesante, otras especificaciones válidas.

Los servidores de aplicación típicamente incluyen el middleware que les permite intercomunicarse con diferentes servicios de forma segura y proporcionan a los desarrolladores un API que aísla a las aplicaciones del sistema operativo del servidor y brindan soporte a una gran variedad de

estándares, tales como HTML, XML, IIOP, JDBC, SSL, etc., que facilitan la interoperatividad de elementos heterogéneos y la conexión a una gran variedad de fuentes de datos, sistemas y dispositivos.

A continuación se abordan algunas de las plataformas que están más extendidas en la actualidad y que proporcionan estos servicios middleware.

4.2.1. Plataforma J2EE

J2EE son las siglas de *Java 2 Enterprise Edition*, es decir, la edición empresarial del paquete Java creada y distribuida por *Sun Microsystems*. Comprende un conjunto de especificaciones y funcionalidades orientadas al desarrollo de aplicaciones empresariales. Debido a que J2EE no deja de ser la definición de un estándar, existen otros productos desarrollados a partir de la misma, aunque no exclusivamente.

Algunas de las funcionalidades más importantes que aporta esta plataforma son las siguientes:

- Proporciona acceso a bases de datos (JDBC).
- Disponibilidad de implementaciones de la plataforma por diferentes fabricantes: BEA, IBM, Oracle, Sun, y Apache Tomcat entre otros.
- Permite la utilización de directorios distribuidos (JNDI).
- Proporciona acceso a la invocación de métodos remotos (RMI, CORBA).
- Dispone de funciones de correo electrónico (JavaMail).
- Aplicaciones Web (JSP y Servlet).
- Puede emplear componentes software como: Beans, objetos CORBA, etc.

La Plataforma J2EE desarrolla el concepto de contenedores a partir de la tecnología *Java 2 Estándar Edition* (J2SE) propuesta también por Sun Microsystems, integra los elementos existentes (JSP, applet, servlet, Javabeans), lo extiende al de *Enterprise Java Bean* (EBJ) como modelo de componentes software distribuidos y basa la comunicación en el estándar RMI (*Remote Methode Invocation*) tratado en temas anteriores.

En la figura 4.7 se presenta un diagrama con los componentes que forman parte de esta plataforma. Se puede observar cómo los diferentes objetos del sistema (JSP, servlet, Applet o EJB) se ejecutan en *contendores* concretos: de aplicación cliente, Web, EJB y navegador Web compatible.

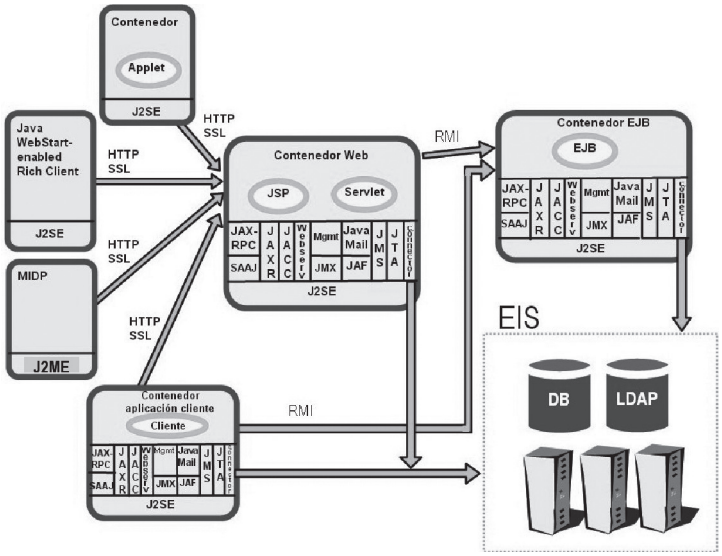


Figura 4.7. Arquitectura J2EE.

Todos los contenedores están basados en J2SE como núcleo básico. Se puede encontrar una excepción en el caso de dispositivos móviles o sistemas embebidos de muy poca capacidad, para los cuales se ha tenido que proponer y desarrollar un núcleo de J2SE mucho más compacto y ligero que se denomina *Java 2 Mobile Edition* (J2ME). Sobre este núcleo, se agregan aquéllos servicios que puedan resultar útiles a las aplicaciones que se ejecuten en cada contenedor (como JAX-RPC, SAAJ, JTA, JAF o JMX; detallados más adelante dentro de este mismo punto) junto con una serie de conectores que facilitan el acceso de las aplicaciones a sistemas de información empresarial (bases de datos, sistemas heredados, etc.).

4.2.1.1. Servicios midlleware en J2EE

J2EE propone el acceso a los servicios que define a través de una interfaz de programación. Las principales APIs de la plataforma J2EE de SUN son las siguientes:

- **JCA** Arquitectura que para interactuar con una variedad de EIS, incluye ERP, CRM y otra serie de sistemas heredados.

- **JDBC** Acceso a base de datos relacionales.
- **JTA** Manejo y la coordinación de transacciones a través de EIS heterogéneos.
- **JNDI** Acceso a información en servicios de directorio y servicios de nombres.
- **JMS** Envío y recepción de mensajes.
- **JMail** Envío y recepción de correo.
- **JIDL** Mecanismo para interactuar con servicios CORBA.

Por supuesto, se dispone de muchos otros APIs orientados a aspectos como: tratamiento de XML, integración con sistemas heredados utilizando Servicios Web, etc.

4.2.1.2. Enterprise JavaBeans

Los EJBs proporcionan un modelo de componentes software distribuido normalizado para el lado del servidor. El objetivo de los Enterprise beans es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, ...) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes nos permite que éstos sean flexibles y, sobre todo, reutilizables.

No hay que confundir a los *Enterprise JavaBeans* con los *JavaBeans*. Los JavaBeans también son un modelo de componentes creado por *Sun Microsystems* para la construcción de aplicaciones, pero no pueden utilizarse en entornos de objetos distribuidos al no soportar nativamente la *invocación a métodos remotos* (RMI).

El API Enterprise JavaBeans (EJB) permite escribir componentes software en Java, lo que les confiere la funcionalidad de portabilidad e independencia de la plataforma, a diferencia de los componentes ActiveX o COM+, propios de los sistemas Windows, o los VCL, asociados a los entornos de desarrollo Delphi, por citar los más conocidos.

Existen tres tipos de EJBs:

- **EJBs de Entidad** (*Entity EJBs*): su objetivo es encapsular los objetos de lado de servidor que almacenan los datos. Los EJBs de entidad presentan la característica fundamental de la persistencia, ya sea gestionada por el propio contenedor (CMP) o por el bean (BMP).

- **EJBs de Sesión** (*Session EJBs*): gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos: «con estado» (*stateful*) o «sin estado» (*stateless*).
- **EJBs dirigidos por mensajes** (*Message-driven EJBs*): los únicos beans con funcionamiento asíncrono. Usando el *Java Messaging System* (JMS), se suscriben a un «tópico» (*topic*) o a una «cola» (*queue*) y se activan al recibir un mensaje dirigido a dicho tópico o cola. No requieren de su instanciación por parte del cliente.

4.2.1.3. Funcionamiento de un Enterprise JavaBean

Los EJBs se disponen en un *contenedor EJB* dentro del *servidor de aplicaciones*. La especificación describe cómo el EJB interactúa con su contenedor y cómo el código cliente interactúa con la combinación del EJB y el contenedor.

Cada EJB debe facilitar una clase de implementación Java y dos interfaces Java. El contenedor EJB creará instancias de la clase de implementación Java para facilitar la implementación EJB. Las interfaces Java son utilizadas por el código cliente del EJB. Las dos interfaces, conocidas como «interfaz home» e «interfaz remota», especifican las firmas de los métodos remotos del EJB.

El servidor invocará a un método correspondiente a una instancia de la clase de implementación Java para manejar la invocación del método remoto. Las llamadas a métodos en la interfaz remota se remiten al método de implementación correspondiente del mismo nombre y argumentos

Dado que se trata simplemente de interfaces Java y no de clases concretas, el contenedor EJB genera clases para esas interfaces que actuarán como intermediarias —o como un «proxy»— entre los clientes y los componentes. El cliente invoca un método en los proxies generados que, a su vez, sitúa los argumentos método en un mensaje y envía dicho mensaje al servidor EJB. Los proxies usan RMI-IIOP para comunicarse con el servidor EJB.

RMI es el mecanismo que permite realizar invocaciones a métodos de objetos remotos situados en distintas (o en la misma) máquinas virtuales de Java, compartiendo así recursos y carga de procesamiento a través de varios sistemas.

La arquitectura RMI puede verse como un modelo de cuatro capas (ver figura 4.8):

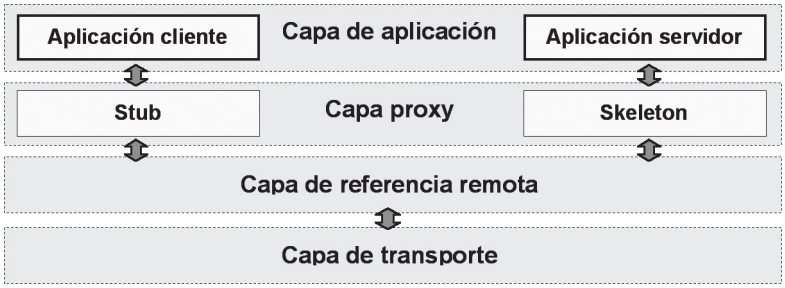


Figura 4.8. Arquitectura RMI en la que J2EE basa la comunicación de sus componentes.

La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. Cualquier aplicación que quiera que sus métodos estén disponibles para su acceso por clientes remotos debe declarar dichos métodos en una interfaz que extienda `java.rmi.Remote`. Dicha interfaz se usa básicamente para *marcar* un objeto como remotamente accesible. Una vez que los métodos han sido implementados, el objeto debe ser exportado. Esto puede hacerse de forma implícita si el objeto extiende la clase `UnicastRemoteObject` (paquete `java.rmi.server`), o puede hacerse de forma explícita con una llamada al método `exportObject()` del mismo paquete.

La segunda capa es la capa proxy, o capa *stub-skeleton*. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.

La tercera capa es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo *stream* (*stream-oriented connection*) desde la capa de transporte.

La cuarta y última capa es la de transporte. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (*Java Remote Method Protocol*), que solamente es válido para aplicaciones Java.

Toda aplicación RMI normalmente se descompone en 2 partes:

- Un servidor, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
- Un cliente, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

4.2.1.4. Servidores de aplicación J2EE

Sun Microsystems, con su servidor de aplicaciones *Sun Java System Application Server,* no es la única que ha desarrollado su especificación J2EE. Bajo «certificados oficiales de compatibilidad», fabricantes diversos han planteado sus propias implementaciones y pueden garantizar que se ajustan completamente al estándar J2EE. *WebSphere* (*IBM*), *Oracle Application Server* (*Oracle Corporation*) y *WebLogic* (*BEA*) están entre los servidores de aplicación J2EE propietarios más conocidos. *EAServer* (*Sybase Inc*.) es también conocido por ofrecer soporte a otros lenguajes diferentes a Java, como PowerBuilder. El servidor de aplicaciones *JOnAS*, desarrollado por el consorcio *ObjectWeb*, fue el primer servidor de aplicaciones libre en lograr certificación oficial de compatibilidad con J2EE. *Tomcat* (*Apache Software Foundation*) y *JBoss* son otros servidores de aplicación libres muy populares en la actualidad.

La portabilidad de Java también ha permitido que los servidores de aplicación J2EE se encuentren disponibles sobre una gran variedad de plataformas, como *Microsoft Windows*, *Unix* y *GNU/Linux*.

4.2.2. Plataforma .NET

.NET es un proyecto de *Microsoft* para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones.

.NET podría considerarse una respuesta de *Microsoft* al creciente mercado de los negocios en entornos Web, como competencia a la plataforma Java de *Sun Microsystems*.

4.2.2.1. .NET Framework

El «framework» o marco de trabajo .NET constituye la base de la plataforma .NET y denota la infraestructura sobre la cual se reúnen un con-

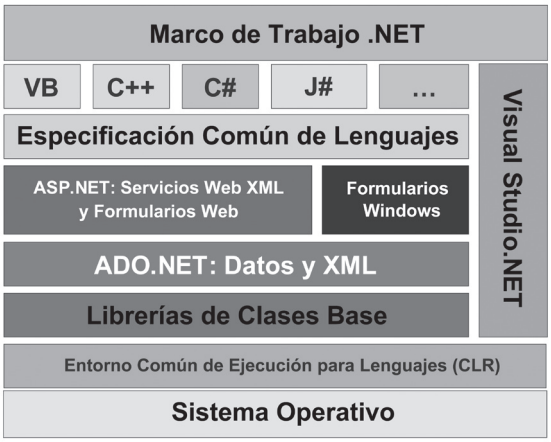


Figura 4.9. Componentes de .NET Framework.

junto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en entornos de ejecución distribuidos. Este marco también reúne una serie de normas impulsadas por varias compañías además de Microsoft (como Hewlett-Packar, Intel, IBM, Fujitsu Software, Plum Hall, la Universidad de Monash e ISE).

Los principales componentes del marco de trabajo son:

- El conjunto de lenguajes de programación.
- La Biblioteca de Clases Base o BCL.
- El Entorno Común de Ejecución para Lenguajes (CLR).

Debido a la publicación de la *norma para la infraestructura común de lenguajes* (*Common Language Infraestructure*, CLI), se ha facilitado enormemente el desarrollo de lenguajes de programación compatibles, por lo que el marco de trabajo .NET soporta ya más de 20 lenguajes de programación y es posible desarrollar cualquiera de los tipos de aplicaciones soportados en la plataforma con cualquiera de ellos, lo que elimina las diferencias que existían entre lo que era posible hacer con uno u otro lenguaje.

Algunos de los lenguajes desarrollados para el marco de trabajo .NET son: *C#*, *Visual Basic*, *Turbo Delphi for .NET*, *C++*, *J#*, *Perl*, *Python*, *Fortran* y *Cobol.NET*.

4.2.2.2. Common Language Runtime

El entorno común para la ejecución (*Common Language Runtime*, CLR) es el verdadero núcleo del marco de trabajo .NET. El CLR es el entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios del sistema operativo.

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .NET en un código intermedio (*Microsoft Intermediate Lenguage*, MSIL), similar al *BYTECODE* de Java. Para generar dicho código el compilador se basa en el *Common Language Specification* (CLS) que determina las reglas necesarias para crear ese código MSIL compatible con el CLR.

Para ejecutarse se necesita un segundo paso. Un compilador JIT (*Just-In-Time*) genera el código máquina real que se ejecuta en la plataforma del cliente; de esta forma se consigue con .NET independencia de la plataforma hardware.

Figura 4.10. Diagrama de la estructura interna del CLR.

La compilación JIT la realiza el CLR a medida que el programa invoca métodos. El código ejecutable generado se almacena en la memoria caché del ordenador, siendo recompilado de nuevo sólo en el caso de producirse algún cambio en el código fuente.

4.2.2.3. Objetos COM+

Bajo las siglas COM+ (*Common Object Model Plus*) se define el modelo de componentes software basados en servicios, propuesto por Microsoft que combina, extiende y unifica los servicios propuestos por los anteriores modelos COM, DCOM y MTS:

- COM (*Component Object Model*). Modelo de objeto componente. Arquitectura software de componentes de Microsoft diseñada para facilitar la reutilización de software y la interoperatividad de los componentes.
- DCOM (*Distributed COM*). Extiende las prestaciones ofrecidas por el modelo COM a través de las redes de computadores.
- MTS (*Microsoft Transaction Server*). Servidor de transacciones de Microsoft. Permite la administración de hilos, gestión de transacciones, operación cooperativa de conexiones de bases de datos y seguridad.

Además de integrar los servicios descritos, añade nuevos modelos: de seguridad, de hilos, administración de transacciones, de administración simplificada; y nuevos servicios: cola de componentes (QC), sucesos débilmente vinculados (LCE), operación cooperativa de objetos y administrador de compensación de recursos (CRM).

El modelo COM+ identifica los siguientes elementos:

- Componente COM: módulo software o archivo binario, por ejemplo los controles ActiveX (generalmente una DLL) o los componentes que encapsulan lógica de negocio en una aplicación distribuida.
- Servidor COM: aplicación que ofrece servicios.
- Cliente COM: aplicación que demanda servicios.
- Interfaz COM: conjunto de métodos y propiedades que definen el comportamiento de un componente y que posibilita el acceso a su funcionalidad.

4.2.2.4. .NET Remoting

.NET Remoting proporciona un marco para que los objetos de distintos dominios de aplicaciones, procesos y equipos se puedan comunicar entre sí con compatibilidad en tiempo de ejecución, permitiendo que estas inte-

racciones se realicen de forma transparente. En un sentido más amplio, también podemos entender *.NET Remoting* como una evolución más de los modelos de componentes antes analizados.

Existen tres tipos de objetos que se pueden configurar como objetos de servicios remotos .NET, y entre los que se puede seleccionar el tipo que mejor se adapte a los requisitos de la aplicación:

- **Objetos Single Call**. Los objetos *Single Call* sólo pueden cubrir una solicitud entrante. Este tipo de objetos resulta bastante útil en aquellos escenarios en los que se debe realizar una determinada cantidad de trabajo y, por lo general, no precisan, ni pueden, almacenar información de estado entre las distintas llamadas a métodos. No obstante, se pueden configurar de forma que permitan el equilibrio de la carga.
- **Objetos Singleton**. Los objetos *Singleton* sirven a varios clientes y, por lo tanto, pueden compartir información al almacenar el estado entre las llamadas de los clientes. Resultan bastante útiles cuando los datos se deben compartir obligatoriamente entre los clientes, y en aquellas situaciones en las que se produce un uso significativo de los recursos derivados de la creación y mantenimiento de los objetos.
- **Objetos activados en el cliente** (CAO). Los objetos activados en el cliente son objetos del lado del servidor que se activan cuando el cliente realiza una solicitud. Este método de activación de los objetos de servidor es muy similar al de la clásica activación de una clase asociada «*CoClass*» de COM.

Los dominios de aplicaciones y las aplicaciones .NET se comunican entre sí a través de mensajes. Los denominados *servicios del canal de .NET* facilitan el medio de transporte necesario para establecer estas comunicaciones.

El marco .NET proporciona dos tipos de canales: los *canales HTTP* y los *canales TCP*, aunque los productos de terceros pueden escribir y conectarse a sus propios canales. El canal HTTP utiliza el protocolo SOAP de forma predeterminada para establecer la comunicación, mientras que el canal TCP recurre a la carga binaria.

Los objetos de servicios remotos .NET se pueden alojar en:

- **Un ejecutable administrado**. Los objetos de servicios remotos .NET se pueden alojar en cualquier archivo .NET EXE normal o en un servicio administrado.

- **Internet Information Server** (IIS). De forma predeterminada, estos objetos reciben los mensajes a través del canal HTTP. Para alojar objetos de servicios remotos en IIS se debe crear una raíz virtual, en la que se copiará un archivo `remoting.config`. El ejecutable o la DLL que contiene el objeto remoto se debe colocar en el directorio `bin`, en el directorio al que señala la raíz de IIS. Se pueden exponer los objetos de servicios remotos .NET como servicios Web.
- **Servicios de componentes .NET**. Los objetos de servicios remotos .NET se pueden alojar en la infraestructura de servicios de componentes .NET para aprovechar los diferentes servicios de COM+, tales como las transacciones, JIT, la agrupación de objetos, etc.

4.2.3. CORBA

CORBA (*Common Object Request Broker Architecture*) es un diseño de middleware que, análogamente a J2EE o a .Net framework, permite que los programas de aplicación se comuniquen unos con otros con independencia de sus lenguajes de programación, sus plataformas hardware y software, las redes sobre las que se comunican, su ubicación y sus implementadores. De hecho, CORBA representa la primera propuesta de middleware aceptado como tal.

Esta arquitectura común para la negociación de solicitudes de objetos fue propuesta por el grupo de administración de objetos OMG. Su principal cometido es trasladar una petición de un objeto cliente —componente CORBA— a una implementación de un objeto, proporcionando los mecanismos por los que los objetos hacen peticiones y reciben respuestas de forma transparente.

Para ello proporciona un modelo de programación sobre bloques básicos arquitectónicos y un nivel de abstracción que permite a las aplicaciones desarrolladas sobre él obtener independencia con respecto a las plataformas sobre las que actúan.

CORBA define los siguientes elementos:

- IDL (*Interface Definition Language*). Lenguaje de definición de interfaces.
- CDR (*CORBA Data Representation*). Representación de datos de CORBA.
- ORB (*Object Request Broker*). Intermediario de petición de objetos.
- IIOP (*Internet Inter ORB Protocol*). Protocolo Inter-ORB para Internet.

- GIOP (*General Inter ORB Protocol*). Protocolo General Inter-ORB.
- RMI (*Remote Method Invocation*). Invocación a métodos remotos.

Las aplicaciones se construyen mediante objetos CORBA, que implementan interfaces definidas en IDL. Los clientes acceden a los métodos de las interfaces de los objetos CORBA mediante RMI. El componente middleware que da soporte a RMI es ORB.

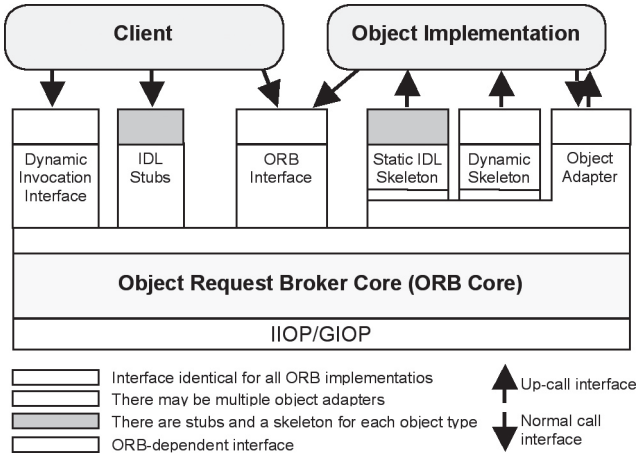


Figura 4.11. Arquitectura CORBA.

Los servicios CORBA proporcionan un equipamiento genérico que puede utilizarse en una gran variedad de aplicaciones. Estos servicios incluyen el servicio de nombres, los servicios de eventos y notificación, el servicio de seguridad, los servicios de transacción y concurrencia y el servicio de comercio.

4.3. INTEGRACIÓN DE TECNOLOGÍAS

Durante estos primeros cuatro capítulos del libro hemos ido desgranando, una tras otra, múltiples tecnologías, herramientas e implementaciones orientadas al desarrollo de grandes aplicaciones distribuidas sobre Internet. El problema es que este tipo de aplicaciones son complejas y, en muchas ocasiones, disponer de un abanico tecnológico tan amplio puede

resultar un verdadero quebradero de cabeza a la hora de contestar a preguntas como: qué tecnología es la más adecuada en cada momento, cómo emplearla de la mejor forma y, sobre todo, cómo trabajar con diferentes propuestas de diferentes fabricantes al mismo tiempo.

Una de las mejores opciones para afrontar este tipo de problemas es aplicar soluciones que ya han demostrado ser exitosas en otros supuestos. Este tipo de soluciones se han ido recogiendo desde los años 60 en especificaciones que denominamos «patrones». Existen patrones en muy diversos ámbitos, como en el diseño, en la optimización o en la organización de las aplicaciones.

En este apartado veremos cómo un patrón de diseño conocido como MVC, aplicado a las diferentes tecnologías que propone J2EE, puede resultar de gran ayuda. Por supuesto, la propuesta es directamente aplicable a otras tecnologías como .NET.

4.3.1. El paradigma MVC

Las tecnologías vistas se enmarcan dentro del desarrollo de aplicaciones Web orientado a objetos y, por lo tanto, deberían responder al *patrón de diseño Modelo-Vista-Controlador* (MVC). Este patrón propone dividir la aplicación en tres partes diferenciadas:

- **El Modelo** o la lógica de negocio de la aplicación. De entre todos los elementos tecnológicos que han ido apareciendo alrededor del modelo Web básico, y que hemos estudiado en capítulos anteriores, el papel de *modelo* queda reservado a los *servidores de aplicación*, gestionados mediante un *marco de trabajo middleware* y basando los desarrollos en el modelo de *componentes software distribuidos* al que estos marcos proporcionan servicios.
- **La Vista** es responsable de gestionar el nivel de presentación, proporcionando una interfaz de usuario altamente desacoplada, tanto de la lógica de negocio como, sobre todo, del cliente. En la práctica, resulta muy aconsejable poder aislar al máximo este apartado del resto de la aplicación, de forma que se pueda dedicar personal especializado en aspectos como el diseño gráfico y de interfaces de usuario más que en el de aplicaciones.
- **El Controlador**. Es el componente que se encarga de manejar la interacción entre la *vista* y el *modelo*, y tiene definido el flujo de la aplicación. Actúa como catalizador necesario para desarrollar

las dos partes anteriores. De su elección depende principalmente que se puedan diferenciar nítidamente los distintos roles que deben establecerse para el desarrollo de aplicaciones Web.

Como se ha comentado ya, la lógica de negocio o *Modelo* deberá estar codificada en forma de componentes software, que la encapsulen y la hagan portable y reutilizable entre diferentes sistemas. Sin embargo, la manera en la que repartimos los otros dos papeles es algo abierto a discusión.

Veamos a continuación las distintas posibilidades utilizando como ejemplo de tecnología: servlets por parte de la ejecución de programas, JSP por parte de las páginas activas y Enterprise JavaBean por parte del middleware y de los componentes. En general, la mayor parte de las reflexiones que aquí se realizan son válidas para otros tándems —por ejemplo, el basado en ASP y COM+ o el basado en Liveware y CORBA, por no mencionar todas las posibilidades de interacción entre todos ellos empleando lenguajes de definición normalizados como XML.

4.3.1.1. Aplicación Web basada en JSP

Utilizando únicamente esta tecnología, la lógica de programación se implementa directamente en las páginas JSP. Las páginas JSP harán el papel de controlador y vista a la vez. Si esto es así, los navegadores deberán hacer peticiones directamente a estas páginas, las cuales interactuarán con los componentes (Beans) necesarios e insertarán los resultados obtenidos en formato HTML, el cual volverá al cliente.

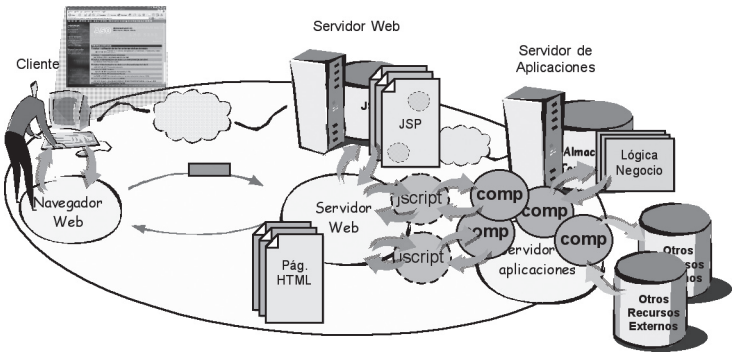


Figura 4.12. Escenario en el que se despliega una aplicación distribuida organizada según un patrón MVC implementado con tecnología JSP.

Las páginas JSP se encuentran, además, perfectamente integradas con el entorno general del servidor de aplicaciones, pudiendo relacionarse con las otras *piezas* que integran el mismo.

Es importante recordar que el código fuente (los *scripts*) incrustado en las páginas JSP se extrae y, a partir del mismo, se generarán sus respectivos objetos *servlets,* que es lo que realmente ejecutará el contendor JSP.

4.3.1.2. Aplicación Web basada en Servlets

En este modelo, los servlets hacen el papel de controlador y de vista a la vez, es decir, contienen la lógica de programación y el resultado o presentación final codificados en Java. Esto implica que cada vez que haya una modificación de formato, aunque no la haya de lógica, debemos modificar y recompilar el servlet correspondiente.

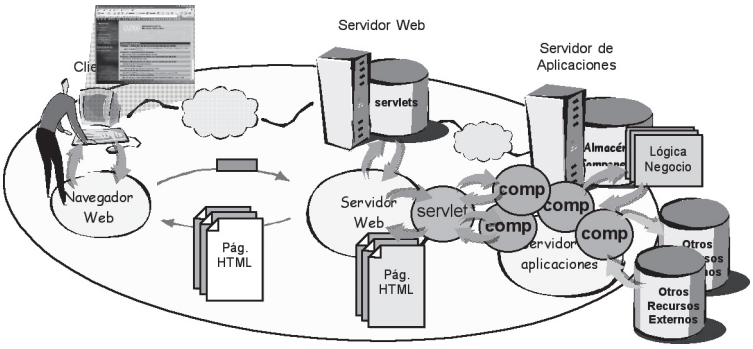


Figura 4.13. Escenario en el que se despliega una aplicación distribuida organizada según un patrón MVC implementado con tecnología *Servlet*.

Aunque aporta un mecanismo muy potente, en la práctica, no facilita la independencia necesaria entre los diferentes papeles de equipo de trabajo que deben confluir en el desarrollo de este tipo de sistemas.

4.3.2. MVC aplicado a J2EE

En este caso, la especificación J2EE fue realizada por SUN para, precisamente, abordar este tipo de problemas inherentes al desarrollo de apli-

caciones distribuidas. Por lo tanto, cuenta con todos los ingredientes para desplegar totalmente el patrón MVC. De hecho, la propia SUN, consciente de la complejidad que entraña la utilización de su plataforma, fue la primera en facilitar a sus usuarios su propia versión sobre cómo aplicar el patrón MVC con J2EE. Esta especificación se conoce como «el modelo 2».

Según el *modelo 2 de SUN*, la lógica de programación irá ubicada en servlets, que posteriormente invocan a páginas JSP. Podemos colocar la lógica de programación en los servlets, de forma que los clientes les invoquen y éstos, a su vez, a páginas JSP, las cuales accederán a la información que necesiten sólo para presentar resultados.

Serán los servlets los que interactuarán con los JavaBeans con el fin de realizar cualquier cálculo o acceder a la lógica de negocio. Los servlets podrán además crear Beans donde almacenar los resultados obtenidos.

Posteriormente, las páginas JSP extraerán la información requerida y almacenada en estos Beans, con el fin de insertarla en su HTML. El resultado final, tras mezclar el contenido dinámico con el estático, se envía al cliente o navegador.

Las páginas JSP constituyen una tecnología mediante la cual podemos presentar páginas con contenido dinámico en base a etiquetas (*tags*) especiales embebidas en el código HTML de la página. Mediante estas etiquetas podremos incrustar código Java para ejecutar lógica de aplicación directamente o bien acceder a un componente externo que realice esta tarea por él, devolviéndole posteriormente un resultado.

Al actuar los servlets como controladores, las páginas JSP, al margen del HTML normal de la página, deben llevar sólo *tags* especiales para su relación con los JavaBeans.

Usando los llamados «*scriplets*» podremos introducir código Java en nuestras páginas JSP. Pero no debemos abusar de esta funcionalidad, ya que cuanto más lo hagamos, más dificultoso será separar los diferentes roles o perfiles en un equipo de desarrollo. En la medida de lo posible, la lógica de programación deberá ir en los servlets, la lógica de negocio en los JavaBeans y la presentación o interfaz gráfica en páginas JSP.

Este último enfoque es el método que juzgamos más idóneo, ya que separa la vista o presentación (en páginas JSP) de la lógica de programación (en servlets) y aporta la indudable ventaja de que podemos separar el contenido de presentación o estático del contenido dinámico, estando la lógica de negocio en componentes o Beans externos a la página.

Según este esquema, en la figura 4.14 podemos apreciar una representación gráfica de la política de trabajo propuesta en este apartado. El clien-

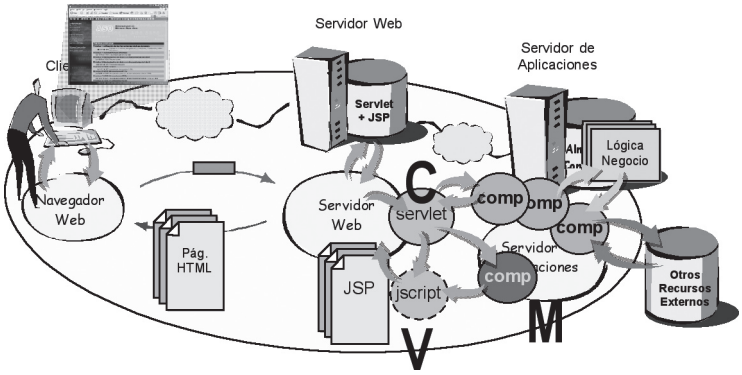


Figura 4.14. Escenario en el que se despliega una aplicación distribuida organizada según un *modelo 2* propuesto por SUN en el que se identifica el (M)odelo, la (V)ista y el (C)ontrolador.

te interactúa con los servlets, en los que va ha estar albergada la lógica y el flujo de programación. Desde ellos se accede a las clases que encapsulan toda la lógica de negocio (*Business Objects*), que son independientes de la filosofía Web.

Los resultados que se obtengan se almacenan en Beans (*View Objects*) y se presentan a través de páginas JSP, las cuales tienen acceso a los mismos para obtener la parte dinámica. Si la presentación es estática, se presentan simples páginas HTML.

Si somos capaces de separar en la medida de lo posible estos dos mundos —presentación y lógica—, estaremos diferenciando también entre el papel del personal dedicado al diseño y a la usabilidad, de aquél dedicado a plasmar el modelo de negocio, lo que redundará en su especialización, pudiendo cada uno de ellos emplear para su cometido las mejores herramientas en cada uno de los campos.

4.3.3. Escenario de desarrollo

En la figura 4.15 se presenta un posible escenario de desarrollo, más o menos complejo, en el que confluyen la mayor parte de los elementos abordados en esta lección.

Por parte de los clientes, podemos observar diferentes tipos de dispositivo y mecanismos de interconexión a la Red desde PCs convencionales,

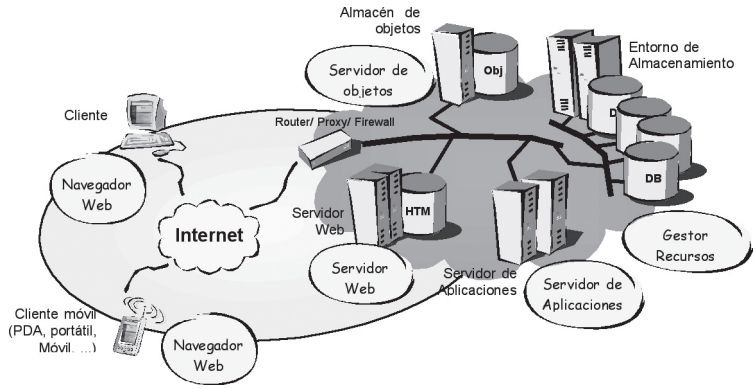


Figura 4.15. Escenario general en el que se recogen los principales componentes físicos y lógicos que proporcionan soporte a una aplicación distribuida sobre Internet.

hasta PDAs o teléfonos móviles. Por parte del servidor Web, se plantea un sencillo cluster (posiblemente soportando balanceo de carga) que representará el punto de entrada a la aplicación; por supuesto, apoyándose desde el punto de vista operativo en los dispositivos típicos de red y seguridad como *routers*, *proxys* o *firewalls*. En lo que respecta a los servidores de aplicaciones, se han separado los nodos de ejecución de la aplicación o de los componentes software de aquellos nodos que pueden servir como almacén de componentes. Finalmente, se proporciona un entorno de almacenamiento tipo NAS (*Network Attached Storage*) o SAN (*Storage Area Networks*), que puede corresponder tanto a aplicaciones y sistemas heredados como a almacenes de persistencia de los propios componentes.

4.4. CASO DE USO: JBOSS

JBoss es un servidor de aplicaciones J2EE de código abierto implementado en Java puro. Al estar basado en Java, JBoss puede ser utilizado en cualquier sistema operativo que soporte la máquina virtual de java. Los principales desarrolladores trabajan para una empresa de servicios, JBoss Inc., adquirida por Red Hat en abril del 2006, fundada por *Marc Fleury*, el creador de la primera versión de JBoss. El proyecto está apoyado por una red mundial de colaboradores. Los ingresos de la empresa están basados en un modelo de negocio de servicios.

JBoss AS fue el primer servidor de aplicaciones de código abierto, preparado para la producción y certificado J2EE 1.4, disponible en el mercado, ofreciendo una plataforma de alto rendimiento para aplicaciones de negocio electrónico (actualmente se está desarrollando la versión para J2EE 5). Combinando una arquitectura orientada a servicios revolucionaria con una licencia de código abierto, JBoss AS puede ser descargado, utilizado, incrustado, y distribuido sin restricciones por la licencia. Por este motivo es la plataforma más popular de middleware para desarrolladores, vendedores independientes de software y, también, para grandes empresas.

Las características destacadas de JBoss incluyen:

- Producto de licencia de código abierto sin coste adicional.
- Cumple los estándares J2EE.
- Confiable a nivel de empresa
- Incrustable, orientado a arquitectura de servicios.
- Flexibilidad consistente
- Servicios del middleware para cualquier objeto de Java
- Ayuda profesional 24x7 de la fuente
- Soporte completo para JMX (*Java Management eXtensions*)

Jboss es un servidor de aplicaciones de uso libre. El grupo Jboss se encarga de la actualización y desarrollo de nuevas versiones y nos permite, si queremos, participar en el proyecto. Jboss se ha convertido en uno de los Servidores de Aplicaciones más usados y con la comunidad más amplia en el mercado. Las distintas distribuciones las podemos encontrar en la página del grupo www.jboss.org. Aquí encontraremos tanto las distribuciones binarias (que será la que se utilice en el caso de uso que se describe) como el código fuente (en caso de que se quisiera colaborar para desarrollar las nuevas versiones).

4.4.1. Instalación

En este caso la distribución binaria que se va a utilizar es `jboss-4.2.0.GA.zip` (última versión estable a fecha junio del 2007) y se puede obtener desde la URL `http://labs.jboss.com/jbossas`.

Para instalarla simplemente se debe descomprimir el paquete binario. La descompresión se hará, por ejemplo, en el directorio local `/home`.

```
# unzip jboss-4.2.0.GA.zip
```

Listado 4.1. Descompresión del servidor de aplicaciones Jboss.

Una vez descomprimido se creará un directorio llamado `jboss-4.2.0.GA` donde se localiza toda la distribución del Servidor Jboss. La estructura de Jboss se comenta a continuación:

- El directorio *bin* contiene los *scripts* de arranque y parada.
- El directorio *server* contiene las configuraciones por defecto del servidor (*all, default y minimal*). En este caso se utilizará la versión *default*. Dentro del directorio *server/default* se localizan los siguientes directorios.
- El directorio *conf* contiene los ficheros de configuración base de la configuración *default* de Jboss
- El directorio *deploy* contiene las aplicaciones propias que queremos ejecutar sobre el servidor de aplicaciones y los servicios ofrecidos por el servidor de aplicaciones en dicha configuración. (Tomcat, mensajería). También contendrá los ficheros de configuración para acceso a bases de datos relacional y los *drivers* JDBC para acceder a ellas. Jboss permite el despliegue de aplicaciones en caliente de tal manera que cuando se añada una aplicación aquí (*ear, war, jar*) Jboss automáticamente la cargará y se podrá acceder a ella (*hot-deploy*).
- El directorio *lib* contiene las librerías necesarias para dicha configuración. Las librerías situadas en esta carpeta se añadirán automáticamente al `classpath`.
- El directorio *log* contiene los archivos de *log* de Jboss. Aunque por consola se obtenga información, es una buena práctica poder almacenarla en archivos de *log*.

4.4.1.1. Inicio y parada

La versión utilizada para el caso de uso lleva incorporado el Servidor Tomcat 6 (de momento no permite la consola de administración de Tomcat) como una aplicación de Jboss completamente integrada.

```
server/default/deploy/jboss-web.deployer
```

En principio cuando se arranque por primera vez el servidor Jboss arrancará el Tomcat como el Servidor independiente que se vio en el tema anterior. Para lanzarlo se utilizará el *script* `run.sh` ubicado en el directorio *bin* de Jboss.

```
run.sh -c configuración_servidor
```

Listado 4.2. Secuencia de arranque del servidor de aplicaciones Jboss.

Al utilizar la configuración *default* no hace falta indicar la opción `-c` para indicar el tipo de configuración de servidor que queremos ejecutar y bastaría lanzar `run.sh`.

Es importante exportar la variable `JAVA_HOME` para que apunte a la máquina virtual de java. En este caso para la distribución utilizada.

```
# export JAVA_HOME=/usr/lib/jvm/java-1.5.0-1.5.0.10
```

Listado 4.3. Variable de entorno para localizar la máquina virtual de Java.

Para detener el Servidor Jboss simplemente se debe utilizar el *script*

```
bin/shutdown.sh
```

Listado 4.4. Secuencia de parada del servidor de aplicaciones Jboss.

4.4.2. Configuración

La configuración del servidor de aplicaciones Jboss es compleja y puede variar en función de las necesidades de la organización. En este apartado se presenta una pequeña parte, y muy básica, de las características que se pueden configurar, como el sistema de *log*, la configuración para utilizar el paradigma de colas y el envío de *emails*.

4.4.2.1. Configuración del sistema de Log

Jboss utiliza el *api Log4j* para generar trazas en la ejecución de aplicaciones. `Log4j` puede enviar errores de distintas maneras: a un archivo, por consola o por mail entre otras.

Por defecto Jboss está configurado para tener dos *appenders* (dos maneras de extraer información de la ejecución de las aplicaciones). El primer *appender* es un archivo situado en el directorio *log* de Jboss de la configuración *default* llamado `server.log`.

El segundo *appender* es la consola la cual se encuentra configurada para que muestre los mensajes de información (`INFO`). La configuración y declaración de los *appenders* se realiza en el archivo `jboss-log4j.xml` ubicado en el directorio *conf* de cada servidor (para este caso *server/default/*). En `log4j` existe una jerarquía de tipos de mensajes que deberán aparecer. Colocándolos de más a menos graves y siempre el menos grave incluirá a los que están por encima de su nivel.

- *OFF* no se muestra en ningún caso.
- *FATAL* para mostrar mensajes de situaciones que probablemente harán abortar la aplicación.
- *ERROR* para mostrar mensajes de errores que no son deseados pero que no interrumpirán la aplicación.
- *WARN* para mostrar mensajes de contextos peligros para la aplicación, o ciertas operaciones de uso no recomendado.
- *INFO* para mostrar mensajes de información sobre la ejecución de la aplicación, o eventos importantes dentro de la misma.
- *DEBUG* para mostrar mensajes interesantes para depurar la aplicación. Muy orientado al tiempo de desarrollo.
- *ALL* se muestra en todos los casos

En el caso que se describe, en la consola, se tiene configurado el estado *INFO* y en entornos de producción se debería tener en estado *WARN* como mucho para que sea más fácil a los administradores localizar los posibles mensajes de error. El *appender* que muestra los mensajes a través de un fichero tendrá por defecto *ALL*, el mínimo, puesto que interesa tener información completa de lo que está sucediendo. El grado de error se le indica como un *elemento xml* `param` dentro de la etiqueta *appender*.

Es posible modificar el nivel de información de *log* cambiando el atributo `value="INFO"` por el nivel que se considere oportuno.

```
<appender name= "CONSOLE" class= "org.apache.log4j.
ConsoleAppender">
<errorHandler class= "org.jboss.logging.util.
OnlyOnceErrorHandler"/>
<param name="Target" value="System.out"/>
<param name="Threshold" value="INFO"/>
..........
</appender>
```

Listado 4.5. Ejemplo de aprender de tipo consola.

Además, no es lógico que el administrador esté pendiente de la consola continuamente y sí es ideal que se notificase de manera automática en caso de producirse un error en el Servidor. Para ello se provee la posibilidad de crear un *appender* que envíe los errores a través del correo electrónico. Para ello, se debe crear un nuevo elemento *appender* debajo del último creado por defecto (cuestión de orden) y darle un nombre en el atributo `name`. En el atributo `class` se le indicará la clase para enviar *emails* (`org.apache.log4j.net.SMTPAppender`). Una vez hecho esto, se especifica el parámetro que dará el nivel de *Log*.

```
<param name=”Threshold” value=”INFO”/>
```

A continuación se muestra otra serie de parámetros para configurar la cuenta a la que debe enviar el correo. Los parámetros se definen como elementos `param` y cambiando sus atributos indicando el nombre del parámetro y su valor.

```
<param name=”nombre_parámetro” value=”valor”/>
```

Parámetros para el correo

- `From` quién lo envía
- `SMTPHost` servidor SMTP
- `Subject` asunto
- `To` a quién lo enviamos

Un ejemplo lo podemos ver a continuación

```
<appender name="SMTP" class="org.apache.log4j.net.SMTPAppender">
  <param name="Threshold" value="WARN"/>
  <param name="From" value="jboss@admin.ua.es"/>
  <param name="SMTPHost" value="smtp.mio.ua.es"/>
  <param name="Subject" value="Error en el Servidor"/>
  <param name="To" value="alumno@alu.ua.es"/>
  <layout class="org.apache.log4j.PatternLayout">
  <param name="ConversionPattern" value="[%d{ABSOLUTE},%c{1}]
%m%n"/>
  </layout>
</appender>
```

Listado 4.6. Ejemplo de aprender de tipo *email*.

Como el coste del envío de mensajes es mucho mayor que cuando las trazas se escriben en un simple archivo o por consola y no queremos perder rendimiento, el *appender* creado se enlazará con otro tipo de *appender* que nos evite esta perdida de rendimiento. Para ello se utilizará el `AsyncAppender`. Se debe notificar al `AsyncAppender` el *appender* que delegará en él. En nuestro ejemplo `SMTPError`.

La creación del *appender* a continuación del anterior sería.

```
<appendername="MailAsync"class="org.apache.log4j.AsyncAppender">
       <appender-ref ref="SMTP"/>
</appender>
```

Listado 4.7. Ejemplo de aprender de tipo AsyncAppender.

Ahora se debe indicar a qué categoría pertenece. No es más que una forma jerárquica de indicar que a un componente se le aplique un tipo de traza. La categoría inicial es `ROOT` la cual se encuentra por encima de los distintos componentes y es la que se utilizará.

Ya existe un *root category* en nuestro archivo por lo que simplemente debemos añadir el *aprender* de correo.

```
<root>
    <appender-ref ref="CONSOLE"/>
    <appender-ref ref="FILE"/>
    <appender-ref ref="SMTP" />
</root>
```

Listado 4.8. Lista de *appenders*.

Una vez configurado, se enviarán las trazas del nivel indicado a través del correo electrónico. Para más información sobre log4j consultar la página:

`http://logging.apache.org/log4j/docs/documentation.html`

4.4.2.2. Configuración del servicio de correo en JBOSS

Jboss facilita el acceso a ciertos recursos de tal manera que, simplemente, se deben configurar en los archivos *xml* correspondientes y, de esta

forma, en las aplicaciones que se realicen se podrá acceder a dichos recursos de una manera sencilla con el fin de realizar ciertas operaciones (envío de correo, acceso a base de datos o envío de mensajes a colas JMS). A continuación se va configurar el envío de correo y el sistema de mensajería JMS (*Java Message Service*).

En primer lugar el archivo donde se configuran los parámetros para poder enviar y recibir correo es:

```
jboss-4.2.0.GA/server/default/deploy/mail-service.xml
```

En este archivo es donde se puede configurar una sesión de JavaMail para poder enviar y leer correos de los servidores determinados. De esta manera se tendrá configurada la cuenta de correo para el usuario que se desee y en la aplicación lo único que se debe hacer es rellenar los campos como si se tratase de un mensaje de correo con cualquier sistema de mensajería gráfico (*Outlook*).

Los campos del `mail-service.xml` que se deben modificar son:

```
<attribute name="User">alu1</attribute>
<attribute name="Password">6tyrd</attribute>
```

Donde se configurarán los datos de nuestro usuario de correo.

```
<property name="mail.pop3.host" value="pop3.mi_dominio"/>
```

servidor *pop3*

```
<property name="mail.smtp.host" value="smtp.mi_dominio"/>
```

servidor *smtp*

```
<property name="mail.from" value="admin@mi_dominio"/>
```

Dirección de nuestro usuario.

4.4.2.3. Configuración de mensajería JMS en JBOSS

Los mensajes JMS es una forma de envió y recepción de mensajes de forma asíncrona (paradigma MOM, *Message Oriented Middleware*). Exis-

ten unos objetos denominados *colas* (*queues*) a los cuales se les puede enviar mensajes, tanto de texto como objetos serializables Java. Una serie de *Listener* denominados MDBs (*Message Driven Beans*), un tipo de EJB orientados a mensajería, estarán escuchando en la *cola* a la espera de que lleguen nuevos mensajes y procesarlos de acuerdo a la implementación para lo que hayan sido creados. Una vez el MDB lee el mensaje de la cola y lo procesa correctamente el mensaje desaparece de la cola. El objetivo de este apartado será crear una *cola* y ver que se ha creado correctamente. Por otra parte, están los *Topic* que a diferencia de las *colas*, permiten que varios MDB "diferentes" se encuentren escuchando a la vez a la espera de recibir mensajes. El mensaje del *Topic* desaparecerá una vez todos los MDBs hayan leído y procesado el mensaje correctamente. En este apartado no describirá la configuración de los *Topic*. El archivo donde se crean las *colas* es:

```
jboss-4.2.0.GA/server/default/deploy/jms/jbossmq-
destinations-service.xml
```

Y al final de éste, dentro de la etiqueta `<server>`, se debe añadir la siguiente información.

```
<mbean code="org.jboss.mq.server.jmx.Queue"
name="jboss.mq.destination:service=Queue,name=ejemploQueue">
       <depends optional-attributename="
DestinationManager">jboss.mq:service=DestinationManager
       </depends>
</mbean>
```

Listado 4.9. Lista de *appenders*.

Aquí simplemente se añade este código y se le da el nombre a la *cola* y Jboss ya se encargará de gestionarla. Cuando se arranque el servidor se podrá observar que crea la *cola* o también se puede ver que se ha creado desde la consola de administración en el apartado *JMS destinations*.

```
http://localhost:8080/jmx-console
```

4.4.3. Despliegue de aplicaciones

El despliegue de aplicaciones en el servidor de aplicaciones Jboss no es uno de los objetivos de este libro. Como se vio en el despliegue de apli-

caciones en Tomcat, existe una estructura predefinida por SUN para las aplicaciones Web. En este caso se puede realizar un paquete del tipo *EAR*. Un archivo EAR es un *Enterprise Archive* con el mismo formato que un archivo JAR o ZIP. Un archivo EAR contiene todos los archivos necesarios para desplegar una aplicación en un servidor de aplicaciones J2EE. Contiene el archivo WAR, el cual esta compuesto por los componentes Web de la aplicación (JSP, Servlets, etc.) y los archivos de tipo JAR, que contienen los componentes de negocio y ciertos archivos de configuración y descripción para el proceso de despliegue. Para más información se puede consultar en la direcciones `http://java.sun.com/j2ee/verified/packaging.html`, `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Overview5.html` o `http://java.sun.com/blueprints/code/projectconventions.html`.

Una vez se tiene el archivo EAR solamente queda ubicarlo en su correspondiente directorio. En este caso en `jboss_home\server\default\deploy`. El servidor de aplicaciones Jboss permite el despliegue en caliente (*hot-deploy*), lo que permite desplegar la aplicación en el directorio correspondiente y no será necesario parar el servidor para poder acceder a la aplicación. Este se encarga de reconocer la aplicación y desplegarla para que el usuario inmediatamente pueda accederá ella.

4.4.4. Integración de Apache, Tomcat y Jboss

Como se vio en el tema anterior, configuración de Tomcat, se puede utilizar un servidor Web especializado para servir páginas estáticas y para poder obtener toda la potencia de éste. En esta configuración Apache se encargaría de las páginas estáticas, Tomcat de los JSP y Servlets y Jboss de los EJBs.

La comunicación se realizará a través del conector *Coyote JK 2* mediante el protocolo *AJP/1.3* Para realizar la comunicación se debe tener en cuenta cuatro detalles.

- Configuración de Tomcat para tener activo el conector (Esta vez incluido en Jboss)
- Obtener e instalar el módulo *mod_jk* de Apache para la comunicación.
- Configuración de Apache para cargar el módulo y configurar la comunicación en su parte.
- Crear un archivo `workers.properties` que usará Apache para obtener los datos de comunicación con Tomcat.

4.4.4.1. Configuración de Tomcat

Al igual que en el tema anterior (apartado 3.4.5.1), se debe configurar el fichero `server.xml` para que esté activo el conector *Coyote JK 2*. Ahora el fichero de configuración del Tomcat se encuentra en:

```
jboss-4.2.0.GA/server/default/jboss-web.deployer/server.xml
```

En esta distribución de Jboss no es necesario realizar ningún cambio, a no ser que se quiera alguna característica más específica. Por defecto, el conector ya viene configurado. El conector se comunica con Apache a través del *puerto 8009*.

4.4.4.2. Instalación del módulo mod_jk

La instalación y configuración del módulo *mod_jk* se realizará de la misma manera que en el apartado 3.4.5.2. Sólo se deben realizar pequeños cambios con respecto a dónde se encuentra ubicado el servidor Tomcat.

4.4.4.3. Configuración de Workers

Un Tomcat *workers* es una instancia de un servidor Tomcat que está esperando que le lleguen peticiones para ejecutar Servlets provenientes de un servidor Web. Para su configuración utilizaremos el mismo `worders.properties` que en tema del servidor Tomcat pero utilizando los directorios apropiados en su configuración. En este caso se debería cambiar el valor de `CATALINA_HOME`.

```
# export CATALINA_HOME=path_de_Jboss/Server/default/deploy/jboss-
web.deployer
```

Listado 4.10. Listado del archivo `worker.properties`

5. COPIAS DE SEGURIDAD

El subsistema de copias de seguridad se ha convertido en un elemento clave en cualquier sistema informático de una organización. Gracias a este subsistema los administradores son capaces de recuperar el estado de dichos sistemas informáticos en un momento dado ante cualquier catástrofe, y en el menor tiempo posible, con el consiguiente beneficio para el funcionamiento de su organización.

En la primera parte de este capítulo veremos los aspectos fundamentales a tener en cuenta a la hora de planificar una correcta política de generación de copias de seguridad: tipos de catástrofes, tipo de información de la que hacer copias, elección de los medios de almacenamiento de las copias, tipos de copias de seguridad… En la segunda parte describiremos las herramientas empleadas tradicionalmente por los administradores para la generación de sus copias de seguridad, generalmente utilizadas dentro de scripts que se ejecutaban periódicamente, con los consiguientes problemas de portabilidad entre sistemas y escalabilidad que presentaban. Por ultimo veremos un caso de uso de sistemas modernos que facilitan el trabajo de los administradores a la hora de generar, almacenar y recuperar las copias de seguridad: Bacula.

5.1. POLÍTICAS DE COPIAS DE SEGURIDAD

A la hora de planificar una política de copias de seguridad ésta debe cumplir, al menos, las siguientes características:

- Fiable: Debemos seleccionar medios de almacenamiento y ubicaciones físicas de los mismos, así como medios de comunicación entre los sistemas que almacenan nuestros datos y el sistema de copias de seguridad, que nos garanticen que nuestras copias se guarden sin pérdidas durante largos espacios de tiempo hasta que necesitemos recuperarlas.
- Distribuido en el tiempo: la Ley de Protección de Datos obliga a que se realice, para el nivel básico de datos de carácter personal, una copia de respaldo al menos semanalmente salvo que no se hayan producido modificaciones en los mismos. Sin embargo, como veremos posteriormente, es recomendable hacer copias con la máxima frecuencia posible (lo aconsejable es que sean diarias) para en caso de perdidas poder devolver nuestros sistemas a un estado estable con la mínima perdida de datos.
- Distribuido en el espacio: la Ley de Protección de Datos también dice que las copias de seguridad deben hallarse en un lugar diferente a donde se encuentren los datos de los que se hacen copias, garantizando así que ante un desastre de carácter físico (fallo hardware del equipo, corrupciones de datos, …) dichas copias no se vean afectadas. Dicha ley no obliga a que las copias se encuentren en otra ubicación física diferente, pudiendo estar en la misma habitación que los equipos de los que se hacen las copias, pero es recomendable, sobre todo en caso de accidentes físicos graves (incendios, inundaciones, …) que dichas copias se almacenen en habitaciones o incluso edificios distintos al de los equipos de los que se hacen las copias. También es recomendable, aunque depende del presupuesto destinado al subsistema de copias de seguridad, que se almacenen las mismas copias en más de una ubicación física (lo ideal es por duplicado o por triplicado, en edificios distintos cada una de estas copias) para minimizar las perdidas en caso de que ocurra un accidente físico en el lugar donde se guardan las copias de seguridad.
- Recuperación rápida y eficiente: Ante una pérdida de datos lo más probable es que nuestra organización, o al menos algún subsistema de la misma, pueda dejar de funcionar correctamente hasta la recuperación de dichos datos. Por lo tanto, debemos seleccionar un sistema de copias de seguridad que nos permita devolver los datos al último estado fiable conocido en el mínimo tiempo posible, con el consiguiente beneficio para el funcionamiento de nuestra organización.

Otro aspecto fundamental que debemos tener en cuenta es la información de la que queremos realizar las copias. Esta debe ser la información crítica para el funcionamiento de la empresa (los archivos de las tablas de nuestra base de datos, las paginas web de nuestro servidor web, los directorios de los usuarios como /home, …) y los de configuración de los equipos (/etc, …). Por ejemplo, no tiene sentido guardar información de los dispositivos o los ejecutables del sistema operativo (por ejemplo, /bin, /usr/sbin, /lib, /proc, …), ya que dicha información ya se encuentra almacenada en los CD's de instalación del sistema operativo y estaríamos gastando recursos innecesariamente.

Como podemos ver, la elección de una política de copias de seguridad no es algo trivial. En dicha elección intervienen una gran variedad de factores, tales como la elección de los datos a almacenar, la ubicación física del lugar de almacenamiento de las copias, elección de la(s) tecnología(s) de almacenamiento para nuestras copias, planificación temporal de las copias, elección de un gestor de copias de seguridad que permita una recuperación rápida y eficiente, presupuesto económico asignado por nuestra organización al sistema de copias de seguridad… En el resto de esta sección vamos a profundizar en algunos de estos aspectos, lo cual nos permitirá realizar una selección más adecuada basada en las necesidades de nuestro sistema de copias de seguridad.

5.1.1. Medios de almacenamiento

A la hora de elegir el medio de almacenamiento debemos tener en cuenta en primer lugar el tamaño de los datos a almacenar y de las copias temporales de los mismos, así como del presupuesto del que disponemos para el sistema de copias de seguridad, intentando buscar una solución de compromiso que se adapte lo máximo posible a dichos requerimientos. También debemos elegir una tecnología que nos permita en un futuro ampliar fácilmente la capacidad de almacenamiento de nuestro sistema en caso de que fuera necesario, y que sea lo más estándar posible de forma que ante un fallo hw del equipo que realiza las copias este sea fácilmente sustituible y sin un gran coste económico (por ejemplo, no cuesta lo mismo cambiar un equipo de última generación con un robot que manipula cintas que un simple PC que trabaje con discos duros). Además, debemos tener en cuenta si necesitamos un medio de almacenamiento que requiera soportes extraíbles (unidades de cinta, CD/DVD, …), los cuáles deberemos extraer del equipo (manual-

mente o mediante un robot) y etiquetar y almacenar correctamente, o nos es suficiente con un sistema con unidades fijas (discos duros, …).

A continuación describiremos las tecnologías de almacenamiento más utilizadas en la actualidad (otras tecnologías, como, por ejemplo, los disquettes o las unidades ZIP, no las describiremos al haber quedado ya obsoletas):

- Cintas magnéticas: Han sido tradicionalmente el medio de almacenamiento más utilizado. Entre sus ventajas se encuentran su alta capacidad de almacenamiento y su bajo coste. Como desventajas tenemos la lentitud de lectura/escritura y que el acceso a los datos es secuencial, de modo que para recuperar un determinado grupo de datos debemos leer todo el contenido de la cinta desde el principio hasta la ubicación de dichos datos.
- CD/DVD: Actualmente, con el descenso de precios de las unidades grabadoras de CD/DVD y de los discos de CD y DVD (escribibles y reescribibles), resultan una opción asequible respecto a las unidades de cinta. Ofrecen una alta capacidad de almacenamiento, bajo coste, alta velocidad y acceso aleatorio a los datos. Sin embargo, presentan como principal problema que la gestión no es totalmente desatendida, pues una vez lleno uno de los discos, se debe hacer manualmente el cambio por uno nuevo, así como el etiquetado y almacenamiento, mientras que con las cintas se puede utilizar uno de los muchos robots existentes en el mercado para tales tareas (aunque de elevado coste).
- Discos duros: Gracias al gran aumento de capacidades de almacenamiento y velocidades así como al descenso de precios (sobre todo UDMA y ATA, pues los SCSI siguen teniendo precios elevados), son la opción más económica y versátil del mercado, comenzando a desbancar a las unidades de cinta. Además, ofrecen un acceso aleatorio muy rápido a la hora de escribir y recuperar los datos.

5.1.2. Niveles de copias de seguridad

Como hemos comentado anteriormente, a la hora de planificar una política de copias de seguridad debemos buscar que esta sea distribuida en el tiempo, realizando las copias periódicamente de forma que optimicemos la cantidad de espacio que ocupan, y que faciliten las posteriores recuperaciones de los datos. Es por ello que surgen los llamados niveles de copias de seguridad, los cuales vienen definidos por la naturaleza de

carácter temporal de los datos de los que se realizan las copias, de modo que una copia de cierto nivel almacena los archivos modificados desde la última copia del nivel inferior. Los niveles de copias de seguridad comienzan a numerarse desde el 0 (copia completa) hasta el nivel 9 (cada vez que se modifica un dato se realiza en ese momento una copia de seguridad de la modificación), aunque en la práctica las copias que se generan suelen llegar hasta el nivel 2 (copia diferencial):

- Copia completa (nivel 0): Consiste en copiar la totalidad de los archivos y directorios de los que queremos almacenar copias de seguridad. La realización de este nivel de copias es bastante sencilla al limitarse a realizar una copia exacta de los datos a guardar, pero tienen como principal inconveniente la gran cantidad de recursos hardware y de tiempo que necesitan, por lo cual sólo se suelen realizar al poner en marcha el sistema de copias de seguridad. Además, debemos tener en cuenta que la recuperación de los datos no es sencilla si las copias las tenemos almacenadas en más de un medio físico (varias cintas, por ejemplo), al tener que leerlos todos para poder realizar la restauración.
- Copia incremental: Consiste en copiar solamente los datos modificados desde la realización de otra copia incremental o total. La principal ventaja de estas copias es que requieren menos tiempo y recursos de almacenamiento que las totales. Por ejemplo, si hace una semana realizamos una copia de nivel 0 en nuestro sistema y deseamos una copia incremental con respecto a él (por lo que sería una copia de nivel 1), deberemos de guardar los ficheros modificados en los últimos siete días. Si la semana siguiente quisiéramos hacer otra copia incremental y quisiéramos optimizar los recursos de almacenamiento, haríamos dicha copia respecto a la incremental anterior en vez de respecto a la total.
- Copia diferencial (nivel 2): Consiste en copiar solamente los datos modificados desde la realización de la última copia incremental. En la práctica, este tipo de copias se realizan de forma diaria. De este modo, si nuestras copias incrementales se realizan semanalmente, tendríamos 7 copias diferenciales entre cada copia incremental, mientras que si nuestras copias incrementales fueran mensuales tendríamos entre 28 y 31 diferenciales entre cada una de ellas. Evidentemente, requerirán menos tiempo y recursos que las totales y las incrementales, pero su principal ventaja es que nos permiten

(combinados con los dos tipos de copias anteriores) devolver el sistema a un estado estable más cercano en el tiempo, con el consiguiente beneficio para el funcionamiento de nuestra organización.

A la hora de restaurar un sistema deberemos tener en cuenta esta jerarquía de niveles de copias de seguridad, de modo que para restaurar completamente un sistema, ante una perdida de todos los datos, deberemos restaurar la copia más reciente de cada nivel, comenzando por el nivel 0. Es decir, deberemos restaurar en primer lugar la última copia de nivel 0 que tengamos almacenada, a continuación restauraremos la última copia de incremental de nivel 1 y si las siguientes incrementales se hicieron sobre la incremental anterior (y por lo tanto no son de nivel 1) seguiremos restaurando en orden todas las incrementales realizadas, y terminaremos restaurando la última copia diferencial.

5.2. HERRAMIENTAS PARA GENERAR COPIAS

En la actualidad existen una gran variedad de herramientas específicas, tanto comerciales como de software libre, para la generación de copias de seguridad (en la última parte del capítulo veremos un ejemplo). Sin embargo, hasta hace poco, e incluso en la actualidad, los administradores programaban sus propios scripts, que se ejecutaban periódicamente, para la generación de las copias de seguridad. Si bien estas soluciones pueden no ser tan completas como los programas específicos de copias de seguridad, tienen la ventaja de no ser un software propietario, pudiendo ajustarlo el administrador a sus necesidades. Por lo tanto, un administrador debería conocer la utilidad que dichas herramientas le pueden ofrecer para generar una solución sencilla para generar copias de seguridad adaptada a sus necesidades, siempre y cuando tenga en cuenta que no llegará a ser tan versátil como una herramienta específica.

5.2.1. Herramientas básicas

A continuación vamos a ver como podemos, mediante una serie de sencillos scripts, generar nuestras copias de seguridad (totales, incrementales y diferenciales) utilizando programas que podemos encontrar en cualquier distribución Unix. En concreto, utilizaremos los comandos find y tar para generar un fichero comprimido con la copia que deseamos, así como el planificador de tareas cron para la planificar la ejecución de las copias y

el comando scp para la copia remota mediante conexión segura a nuestro espacio de almacenamiento de las copias de seguridad. A modo de ejemplo, supondremos que queremos almacenar del equipo llamado `maquina1.mired.dtic.ua.es` los cambios realizados en los directorios de los usuarios (/home), y que utilizaremos como servidor de almacenamiento de las copias de seguridad el equipo llamado `copias1.mired.dtic.ua.es`, en el cuál almacenaremos dichas copias en el directorio `/copias/maquina1`.

En primer lugar crearemos, en el equipo `maquina1.mired.dtic.ua.es`, el script que nos genere la copia total (nivel 0) con la que iniciaremos nuestro sistema de copias de seguridad. Dicho script consistirá simplemente en comprimir el contenido de dichos directorios con el comando tar y enviarlo mediante una conexión segura, con el comando scp, a nuestro servidor de copias de seguridad. Evidentemente, para poder realizar dicha copia remota el servidor de copias deberá ejecutar el servidor ssh y estar configurado para permitir las conexiones de ese tipo desde el equipo `maquina1.mired.dtic.ua.es`, y en nuestro caso utilizaremos el usuario `userCopias` del equipo `copias1.mired.dtic.ua.es` para realizar dichas copias.

```
#!/bin/bash

dirDestino='/copias/$HOSTNAME'
comando=`mkdir $dirDestino'

tar cvfpz copiaNivel0.tgz /home
ssh userCopias@copias1 $comando
scp copiaNivel0.tgz userCopias@copias1.mired.dtic ua.es:$dirDestino

rm copiaNivel0.tgz
```

Listado 5.1. Script que genera la copia total (nivel 0).

De este modo ya tenemos almacenado en nuestro servidor de copias la copia de nivel 0 (copia total) a partir de la cual generaremos las incrementales y diferenciales. Como podemos ver en el script, hemos utilizado la variable global $HOSTNAME para crear e indicar el subdirectorio adecuado para almacenar las copias del equipo `maquina1.mired.dtic.`

`ua.es`, permitiendo utilizar este script en cualquier equipo de nuestra red sin necesidad de modificarlo, de forma que las copias para los otros equipos se almacenarían en los subdirectorios adecuados. También borramos la copia generada en la máquina de origen, para no dejar archivos innecesarios en el sistema.

A continuación generamos el script que generará las copias incrementales, las cuales las realizaremos semanalmente. Para ello utilizaremos el comando find, para obtener los ficheros modificados durante los últimos 7 días, y los almacenaremos en el directorio apropiado de la máquina que almacena las copias de seguridad, añadiendo al nombre de dicha copia la fecha de su creación, para facilitar así su futura recuperación, quedando dichos archivos con `copiaInc_Año-Mes-Dia` como formato de nombre.

```
#!/bin/bash

fecha=`date +%F`
nombreCopia="copiaInc_$fecha.tgz"
dirDestino='/copias/$HOSTNAME'

find /home -ctime -7 > listaFicheros.txt
tar cvfpz $nombreCopia -T listaFicheros
scp $nombreCopia userCopias@copias1.mired.dtic.ua.es:$dirDestino

rm $nombreCopia
rm listaFicheros.txt
```

Listado 5.2. Script que genera las copias incrementales semanales.

Además, para ejecutar semanalmente dicho script, al que llamaremos copiaIncremental.sh, deberemos crearlo en el directorio /etc/cron.weekly.

Por último generamos el script que almacenará las copias diferenciales diariamente. Este script, al que llamaremos copiaDiferencial.sh, es muy similar al que genera las incrementales, variando solo el período de búsqueda con el que ejecutamos el comando find y que los archivos los nombraremos siguiendo el formato `copiaDif_Año-Mes-Dia`. Este script se deberá crear en el directorio /etc/cron.daily para garantizar su ejecución diaria.

```
#!/bin/bash

fecha=`date +%F`
nombreCopia="copiaDif_$fecha.tgz"
dirDestino='/copias/$HOSTNAME'

find /home -ctime -1 > listaFicheros.txt
tar cvfpz $nombreCopia -T listaFicheros
scp $nombreCopia userCopias@copias1.mired.dtic.ua.es:$dirDestino

rm $nombreCopia
rm listaFicheros.txt
```

Listado 5.3. Script que genera las copias diferenciales diarias.

5.2.2. Utilidad Rdist

Rdist es un programa de código abierto, que tal y como lo definen sus creadores, nos permite mantener copias sincronizadas de archivos entre equipos. Por lo tanto, puede ser una buena opción si buscamos tener una máquina que pueda actuar de respaldo de otra al estar sincronizados sus datos mediante rdist.

Para realizar las copias rdist nos ofrece la posibilidad de utilizar tanto rsh (el que utiliza por defecto) como ssh, por lo que utilizaremos el segundo, al ofrecernos una conexión segura. Por lo tanto la máquina en la que se realicen las copias deberá tener en funcionamiento un servidor ssh configurado para permitir las conexiones desde la máquina de origen de los datos. También nos ofrece una gran cantidad de opciones de configuración que se le pasan como parámetro, que nos permiten ajustar las características de las copias a generar (omisión de ficheros a copiar, sobreescritura, renombrado de ficheros, …).

Además, para configurar la ejecución de las copias utiliza un archivo que se le pasará como parámetro al invocar el comando rdist, en el que se definen los archivos/directorios a copiar, los equipos en los que se realizarán las copias, así como las características de dichas copias. En nuestro caso, para que la máquina `copias1.mired.dtic.ua.es` tenga una copia sincronizada del directorio /home con respecto al equipo `maquina1.mired.dtic.ua.es`, nuestro fichero de configuración de `maquina1`, al que llamaremos /etc/confRdist, quedará:

```
FILES = ( /home )
HOSTS = ( userCopias@copias1.mired.dtic.ua.es )
${FILES} -> ${HOSTS} install –oremove
             notify userCopias@mired.dtic.ua.es
```

Listado 5.4. Archivo de configuración de Rdist.

Como podemos ver, hemos indicado en la sección HOSTS que utilizaremos el usuario `userCopias` de la máquina `copias1`, así como también indicamos en las opciones de ejecución, mediante –oremove, que la sincronización sea total, es decir, actualizando incluso los borrados de archivos/directorios, y mediante la opción notify hacemos que envié un correo con los resultados de la sincronización al usuario userCopias de nuestro servidor de correo. Por último, nos queda lanzar la ejecución de rdist utilizando el fichero de configuración que acabamos de crear, y utilizando ssh para realizar el copiado. Además, en nuestro caso, como queremos que la sincronización se haga diariamente, deberemos invocar el comando rdist con dichos parámetros desde el sistema cron, para lo cual creamos el script /etc/cron.daily/copiaSegRdist.sh con el siguiente código:

```
#!/bin/bash

rdist –f /etc/confRdist –P /usr/bin/ssh
```

Listado 5.5. Script /etc/cron.daily/copiaSegRdist.sh que lanza la ejecución diaria de rdist.

5.2.3. Utilidad Rsync

Rsync, al igual que rdist, es un programa de código abierto enfocado a la sincronización de archivos y directorios. Además de permitir la comunicación mediante conexión segura con ssh, la principal ventaja que ofrece sobre Rdist reside en su algoritmo de sincronización de los ficheros modificados, el cuál solo transmite las porciones modificadas del archivo en lugar de enviarlo entero, con la consiguiente reducción en el uso del ancho de banda de nuestra red.

Rsync permite una gran variedad de opciones a la hora de su ejecución. Dispone de un demonio rsync que se ejecutaría en la máquina en la que haríamos las copias, pero en la configuración que vamos a utilizar copiaremos los ficheros directamente con el cliente rsync sobre ssh, siguiendo la misma filosofía que en el caso mostrado para rdist, pues en la ultima parte de este capítulo veremos Bacula, un sistema cliente/servidor para copias de seguridad más completo aún que rsync.

Como en el caso anterior, vamos a mantener sincronizado el directorio /home de la maquina `copias1.mired.dtic.ua.es` con respecto al de la máquina `maquina1.mired.dtic.ua.es`, para lo cuál generaremos en esta última el script /etc/cron.daily/copiaSegRsync, la cual se ejecutará diariamente mediante el demonio cron.

```
#!/bin/bash

rsync  -arv -delete -force -rsh="/usr/bin/ssh" "/home" \
"copias1.mired.dtic.ua.es:/home" > resultadoRsync.txt
Mail userCopias@mired.dtic.ua.es -s  \
"sincronizacion Rsync" < resultadoRsync.txt
```

Listado 5.6. Script /etc/cron.daily/copiaSegRsync.sh que lanza la ejecución diaria de rsync.

Mediante las opciones –arv realizamos una copia recursiva de todo el directorio, y generamos una salida detallada que posteriormente enviaremos por correo, mediante el comando Mail, a la cuenta de correo del usuario userCopias de nuestro servidor de correo. Además, con la opción –delete nos aseguramos de borrar en el destino los archivos que no existan en el origen.

5.2.4. Realización de las copias mediante una conexión segura

En los apartados anteriores hemos comentado que utilizaríamos ssh para garantizar la transmisión de los datos por una conexión segura. Sin embargo, ssh por defecto pide contraseña de usuario, a no ser que se le pase como parámetro, no siendo muy recomendable en cuanto a la seguridad el dejar claves de usuario en texto plano en dichos scripts. Por ello, debemos utilizar la autenticación basada en host/usuario, de modo que se permite conectarse al servidor ssh a un determinado usuario desde un equipo concreto.

En primer lugar el cliente, en nuestro caso el usuario userCopias del equipo `maquina1.mired.dtic.ua.es`, deberá generar sus claves privada y pública, compartiendo esta última con el servidor para poder identificarse, evitando así el tener que utilizar la contraseña del usuario para poder conectarse al servidor:

```
maquina1.mired.dtic.ua.es# ssh-keygen –t dsa
```

El programa nos pedirá una passphrase, que en nuestro caso dejaremos vacía para que no nos la pida a la hora de hacer la identificación. De este modo, tendremos almacenada en el fichero /home/userCopias/.ssh/id_dsa.pub la clave pública del usuario userCopias del equipo `maquina1.mired.dtic.ua.es`, que deberemos compartir con el equipo `copias1.mired.dtic.ua.es`, añadiéndolo al archivo /home/userCopias/.ssh/.authorized_keys2:

```
maquina1.mired.dtic.ua.es# ssh userCopias@copias1.mired.dtic.ua.es \
'cat >> .ssh/authorized_keys2' < .sshd/id_dsa.pub
```

Una vez realizada dicha copia de la clave privada, el usuario userCopias del equipo `maquina1.mired.dtic.ua.es` ya podrá acceder por ssh como usuario userCopias al equipo `copias1.mired.dtic.ua.es`, y por lo tanto, los scripts comentados anteriormente para la generación de copias de seguridad utilizando ssh funcionarán correctamente.

5.3. CASO DE USO: BACULA

Bacula es un sistema cliente/servidor formado por un conjunto de programas que permiten administrar la generación, recuperación y verificación de las copias de seguridad de una red de computadores, pudiendo utilizar distintos tipos de almacenamiento, incluyendo cintas, discos duros, unidades de disco y cd/dvd.

5.3.1. Características de Bacula

Bacula tiene un diseño modular, dividido en los siguientes componentes:

- **Bacula Director:** Es el programa que supervisa todas las operaciones de almacenamiento, restauración y verificación de las copias de segu-

ridad. Mediante el Bacula Director el administrador del sistema puede planificar las copias y recuperar archivos a partir de las mismas.

- **Bacula Console:** Es el programa con el cual nos conectaremos al director, y desde donde podremos dar ordenes, hacer consultas, etc… Existen tres versiones, una ejecutable desde el shell, y otras dos versiones gráficas ejecutables desde XWindows (una basada en GNOME y otra en wxWidgets) pero que no incluyen todas las capacidades de la primera.
- **Bacula File:** Es el cliente que se ejecuta como demonio en cada máquina de las que queramos hacer respaldo. Su función es leer y transmitir a un Bacula Storage los ficheros que el Bacula Director le indique para realizar la copia/restauración. Cada versión es específica del sistema operativo sobre el que se ejecuta, habiendo versiones para Unix/Linux y Windows NT/2000/XP/2003/98/Me.
- **Bacula Storage:** Es el programa encargado de la lectura/escritura física de las copias en los volúmenes físicos que estén definidos (cintas, ficheros, cds, …). Se ejecuta como demonio en la máquina en la que se encuentra el dispositivo físico donde se almacenan las copias.
- **Bacula Catalog:** Es la base de datos donde el director guarda y registra todas sus operaciones. Actualmente soporta tres tipos de bases de datos: Mysql, PostgreSQL y SQLite.

Para poder realizar copias y recuperaciones de las mismas deben estar configurados correctamente y ejecutándose como demonios los siguientes componentes del sistema: Bacula Director, Bacula File, Bacula Storage, y Bacula Catalog. De este modo, el Bacula Director ejecutará los trabajos que tiene planificados, comunicándose con los Bacula File correspondientes de los clientes para la transmisión de los archivos necesarios según el tipo de copia/recuperación/verificación que se haya definido. A su vez, cada Bacula File, siguiendo las instrucciones que le indique el Bacula Director, enviará/recibirá los ficheros a copiar/restaurar al Bacula Storage para que actualice los cambios en los dispositivos físicos. Además, el Bacula Director registrará todas estas operaciones, así como el estado de los dispositivos físicos en el Bacula Catalog.

Además, Bacula utiliza un esquema incremental para definir los dispositivos que se utilizarán a la hora de almacenar las copias. Define los volúmenes como los dispositivos físicos donde se guardan las copias (ficheros, discos, cintas…) y los pools como el conjunto de uno o varios volúmenes. De este modo, podemos utilizar un gran número de dispositivos (los volú-

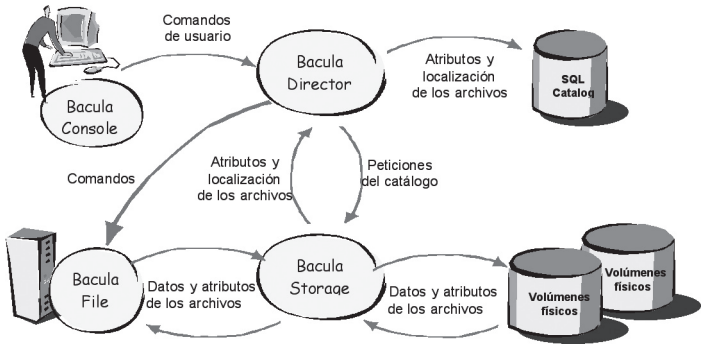


Figura 5.1. Interacción de los componentes de Bacula.

menes), incluso de distinto tipo, para almacenar las copias en el pool que definamos. Además, toda la información sobre los volúmenes y pools, así como las operaciones que se realicen sobre ellos y su estado se mantiene centralizada en el catálogo, lo cual permite la consulta de la misma de una gran variedad de formas: el Bacula Console, clientes de BBDD, aplicaciones web que consulten la BBDD…

5.3.2. Instalación y configuración

Como ejemplo de uso de Bacula, vamos a configurar las copias de seguridad de dos servidores. Vamos a suponer que el primero de estos servidores, `servWeb.dtic.ua.es` es un servidor web que ejecuta Apache, y del cuál queremos guardar copias de seguridad del directorio donde se alojan sus páginas, en concreto /var/www, y sus ficheros de configuración en el directorio /etc/apache2. El segundo de estos servidores, `servBD.dtic.ua.es` es un servidor que ejecuta una base de datos mysql, por lo que estamos interesados en guardar copias de seguridad del directorio /var/lib/mysql, que es donde se almacenan los datos, y del directorio /etc/mysql que contiene su configuración. Dichas copias de seguridad se almacenarán en el servidor de copias de seguridad `servCopias.dtic.ua.es`.

En primer lugar debemos instalar en los servidores de los que queremos hacer copias el demonio Bacula File mediante:

```
servWeb.dtic.ua.es# apt-get install bacula-fd
servBD.dtic.ua.es#  apt-get install bacula-fd
```

El archivo de configuración de cada uno de estos Bacula File es:

```
# List Directors who are permitted to contact this File daemon
Director {
  Name = servCopias-director
  Password = "password"
}

# "Global" File daemon configuration specifications
FileDaemon {                          # this is me
  Name = servWeb-fd
  FDport = 9102         # where we listen for the director
  WorkingDirectory = /var/lib/bacula
  Pid Directory = /var/run/bacula
  Maximum Concurrent Jobs = 20
}

# Send all messages except skipped files back to Director
Messages {
  Name = Standard
  director = servCopias-director = all, !skipped
}
```

Listado 5.7. Archivo de configuración, /etc/bacula/bacula-fd.conf, del Bacula File del servidor servWeb.dtic.ua.es.

```
# List Directors who are permitted to contact this File daemon
Director {
  Name = servCopias-director
  Password = "password"
}

# "Global" File daemon configuration specifications
FileDaemon {                          # this is me
  Name = servBD-fd
  FDport = 9102            # where we listen for the director
  WorkingDirectory = /var/lib/bacula
  Pid Directory = /var/run/bacula
  Maximum Concurrent Jobs = 20
}

# Send all messages except skipped files back to Director
Messages {
  Name = Standard
  director = servCopias-director = all, !skipped
}
```

Listado 5.8. Archivo de configuración, /etc/bacula/bacula-fd.conf, del Bacula File del servidor servBD.dtic.ua.es.

Como podemos ver, en dichos archivos hemos definido tanto el director que podrá conectarse a nuestros Bacula File para ordenarnos trabajos, como el nombre de del propio Bacula File y el puerto por el que se realizará dicha comunicación, así como el tipo de mensajes que se podrán enviar al director.

A continuación vamos a instalar y configurar el Bacula Director, el Bacula Storage y el Bacula Catalog en el equipo servCopias.dtic.ua.es, pero debemos tener en cuenta que podríamos haber instalado cada uno de estos componentes en un equipo distinto siempre que el Bacula Storage se encontrara en el equipo que gestiona los medios físicos en los que se almacenarán las copias de seguridad, pero por simplicidad hemos decidido centralizar dichos servicios en un único equipo. Además, el Bacula Director ofrece tres versiones según el tipo de base de datos que vayamos a utilizar para almacenar el catálogo (mysql, postgres y sqlite) y nosotros vamos a utilizar la basada en mysql. Por lo tanto, para instalar dichos componentes de Bacula debemos ejecutar:

```
servCopias.dtic.ua.es# apt-get install bacula-sd
servCopias.dtic.ua.es# apt-get install bacula-director-mysql
```

A la hora de elegir la ubicación del catálogo mientras instalamos el Bacula Director, seleccionaremos la maquina local (localhost), por lo cual deberemos haber instalado previamente el servidor mysql de bases de datos, aunque podríamos haber elegido cualquier otro equipo que dispusiera de dicho servidor siempre que tuviéramos acceso como cliente a la base de datos de ese equipo (por ejemplo, el equipo `servBD.dtic.ua.es`), pues al realizar dicha instalación se crearán la base de datos y las tablas que componen el catálogo de Bacula.

También debemos indicar la existencia de un bug en el script de postinstalación del director en su versión 1.36.2 para mysql que impide la correcta instalación del mismo. Dicho problema se soluciona comentando en el archivo /var/lib/dpkg/info/bacula-director-mysql.postinst las líneas:

```
# echo -e "GRANT ALL privileges ON $CATALOG.* TO  \
# $MYSQL_USER@$MYSQL_HOST_URI " \
# "$MYSQL_USER_PSWD_STRING;\nFLUSH PRIVILEGES;" \
# | $MYSQL -h $MYSQL_HOST -u $MYSQL_ROOT_USER   \
# $MYSQL_PSWD_STRING mysql
```

Listado 5.9. Líneas a comentar en el archivo /var/lib/dpg/info/bacula-director-mysql.postinst para la correcta instalación del Bacula Director.

Tras comentar dichas líneas volveremos a ejecutar la orden de instalación del Bacula Director:

```
servCopias.dtic.ua.es# apt-get install bacula-director-mysql
```

Una vez instalados los paquetes necesarios, vamos a proceder a modificar los archivos de configuración de dichos componentes de Bacula para adaptarlos al ejemplo que proponíamos.

En primer lugar vamos a configurar el Bacula Storage, indicando que utilice como dispositivo de almacenamiento el directorio /copias, para el cual el usuario bacula del sistema debe tener permisos de acceso y creación de archivos, del propio equipo servCopias.dtic.ua.es, aunque Bacula permite indicar una gran variedad de dispositivos, tales como unidades de cinta, DVDs, … También deberemos indicarle el director desde el cuál recibirá las peticiones y el tipo de mensajes que enviará. Para ello, deberemos modificar el fichero /etc/bacula/bacula-sd.conf como se muestra a continuación:

```
Storage {                                # definition of myself
  Name = servCopias-sd
  SDPort = 9103                          # Director's port
  WorkingDirectory = "/var/lib/bacula"
  Pid Directory = "/var/run/bacula"
}

# List Directors who are permitted to contact Storage daemon Director {
  Name = servCopias-director
  Password = "password"
}

# Devices supported by this Storage daemon. To connect, the
# Director's bacula-dir.conf must have the same Name and MediaType.
Device {
  Name = FileStorage
  Media Type = File
  Archive Device = /copias
  LabelMedia = yes;        # lets Bacula label unlabeled media
  Random Access = Yes;
  AutomaticMount = yes;    # when device opened, read it
  RemovableMedia = no;
  AlwaysOpen = no;
}

# Send all messages to the Director
Messages {
  Name = Standard
  director = servCopias-director = all
}
```

Listado 5.10. Fichero de configuración, /etc/bacula/bacula-sd.conf, del Bacula Storage.

A continuación procederemos a modificar el archivo de configuración, /etc/bacula/bacula-dir.conf, del Bacula Director del equipo `servCopias`.

```
Director {                            # define myself
  Name = servCopias-director
  DIRport = 9101          # where we listen for UA connections
  QueryFile = "/etc/bacula/scripts/query.sql"
  WorkingDirectory = "/var/lib/bacula"
  PidDirectory = "/var/run/bacula"
  Maximum Concurrent Jobs = 2
  Password = "password"   # Console password
  Messages = Daemon
}
Job {   Name = servWeb-mensual
        Client = servWeb-fd
        Type = backup
        Level = Incremental
        FileSet = dirsServWeb
        Schedule = Mensual
        Messages = Standard
        Pool = PoolservWeb-Mensual
        Storage = dirCopiasWeb
   }
Job {
        Name = servWeb-restore
        Client = servWeb-fd
        Type = Restore
        FileSet = dirsServWeb
        Pool = PoolservWeb-Mensual
        Storage = dirCopiasWeb
        Messages = Standard
}
Job {
        Name = servBD-mensual
        Client = servBD-fd
        Type = backup
        Level = Incremental
        FileSet = dirsServBD
        Schedule = Mensual
        Messages = Standard
        Pool = PoolservBD-Mensual
        Storage = dirCopiasBD
}
Job {
        Name = servBD-restore
        Client = servBD-fd
        Type = Restore
        FileSet = dirsServBD
        Pool = PoolservBD-Mensual
        Storage = dirCopiasBD
        Messages = Standard
}
```

Listado 5.11. Primera parte del archivo de configuración /etc/bacula/bacula-dir.conf.

`dtic.ua.es`. Definimos en primer lugar el propio director, indicando cuantos trabajos se podrán ejecutar concurrentemente, así como la contraseña y el puerto que utilizará. A continuación indicamos los dos trabajos a realizar, y sus dos trabajos asociados de tipo Restore para realizar la recuperación de las copias: el del equipo `servWeb.dtic.ua.es`, y el del equipo `servBD.dtic.ua.es`, en los cuales especificamos el nombre del Bacula File del cliente, el tipo de trabajo, el planificador asociado que le indicará la hora de hacerlo, el pool y el storage en el que se almacenará, los cuales se definirán al final del archivo. En nuestro caso hemos definido el trabajo de tipo backup, para que almacene los datos, y el nivel incremental.

En la segunda parte del archivo definiremos los FileSet y el Schedule que planificará los trabajos. En los FileSet indicamos los ficheros a guar-

```
FileSet {
  Name = "dirsServWeb"
  Include {
           Options {
                    Compression = GZIP
                    signature = MD5
                   }
           File = "/var/www"
           File = "/etc/apache2"
           }
}


FileSet {
  Name = "dirsServBD"
  Include {
           Options {
                    Compression = GZIP
                    signature = MD5
                   }
           File = "/var/lib/mysql”
           File = "/etc/mysql"
           }
}


Schedule {
  Name = "Mensual"
  Run = Full 1st sun at 01:05
  Run = Differential 2nd-5th sun at 01:05
  Run = Incremental mon-sat at 01:05
}
```

Listado 5.12. Segunda parte del archivo de configuración, /etc/bacula/bacula-dir.conf, del Bacula Director de servCopias.dtic.ua.es.

dar, así como opciones de compresión y codificación. En el Shedule Mensual definimos que hacemos una copia total (nivel 0) a principios de mes, a final de cada semana realizaremos una diferencial, y por último, cada día una incremental. También debemos tener en cuenta que al hacer la primera copia, aunque sea diferencial o incremental, en realidad hará una total, pues no encontrará una total anterior.

En la tercera parte del archivo vamos a definir los clientes, cuyos nombres deberán coincidir con los especificados en los Bacula Client de los

```
Client {
        name = servWeb-fd
        Address = servWeb.dtic.ua.es
        FDPort = 9102
        Catalog = MyCatalog
        Password = "password"
        File Retention = 30 days    # 30 days
        Job Retention = 6 months    # six months
        AutoPrune = yes             # Prune expired Jobs/Files
}

Client {
        name = servBD-fd
        Address = servBD.dtic.ua.es
        FDPort = 9102
        Catalog = MyCatalog
        Password = "password"
        File Retention = 30 days    # 30 days
        Job Retention = 6 months    # six months
        AutoPrune = yes             # Prune expired Jobs/Files
}

Storage {
  Name = servCopias-sd
  Address = servCopias.dtic.ua.es
  SDPort = 9103
  Password = "password"
  Device = FileStorage
  Media Type = File
}

# Generic catalog service
Catalog {
  Name = MyCatalog
  dbname = bacula;
  DB Address = localhost;
  user = bacula;
  password = "password"
}
```

Listado 5.13. Tercera parte del archivo de configuración, /etc/bacula/bacula-dir.conf, del Bacula Director de servCopias.dtic.ua.es.

equipos correspondientes. A continuación definimos el Bacula Storage que se utilizará para almacenar las copias, indicando su nombre, dirección, puerto y clave, así como el dispositivo y el tipo del mismo. También indicamos el Bacula Catálog que se utilizará para almacenar todas las operaciones y el estado de las copias.

En la cuarta y última parte del fichero definiremos el tipo y cantidad de mensajes que queremos generar y almacenar. Por último, indicaremos en los pools que se nos almacenen copias por un período de 3 meses y como etiquetaremos dichas copias mediante el campo LabelFormat, en nuestro caso siguiendo el formato nombreEquipoYYYYMMDD.

```
Messages {
  Name = Standard
  console = all, !skipped, !saved
  append = "/var/lib/bacula/log" = all, !skipped
}

Pool {
  Name = PoolservWeb-Mensual
  Pool Type = Backup
  Recycle = yes
  Use Volume Once = yes
  AutoPrune = yes
  Volume Retention = 3 month
  Accept Any Volume = yes
  LabelFormat = "servWeb${Year}${Month:p/2/0/r}${Day:p/2/0/r}"
}


Pool {
  Name = PoolservBD-Mensual
  Pool Type = Backup
  Recycle = yes
  Use Volume Once = yes
  AutoPrune = yes
  Volume Retention = 3 month
  Accept Any Volume = yes
  LabelFormat = "servBD${Year}${Month:p/2/0/r}${Day:p/2/0/r}"
}
```

Listado 5.14. Cuarta parte del archivo de configuración, /etc/bacula/bacula-dir.conf, del Bacula Director de servCopias.dtic.ua.es.

A continuación reiniciaremos los demonios asociados a cada uno de los componentes de Bacula en cada uno de los equipos correspondientes, de modo que ya tendremos en marcha nuestro sistema de copias de seguridad:

```
servWeb.dtic.ua.es# /etc/init.d/bacula-fd restart
servBD.dtic.ua.es#  /etc/init.d/bacula-fd restart
servCopias.dtic.ua.es# /etc/init.d/mysql restart
servCopias.dtic.ua.es# /etc/init.d/bacula-sd restart
servCopias.dtic.ua.es# /etc/init.d/bacula-sd restart
servCopias.dtic.ua.es# /etc/init.d/bacula-sd restart
```

5.3.3. Utilización de la consola de Bacula

A continuación vamos a ver como utilizar la consola de Bacula para monitorizar el estado de las copias y realizar la recuperación de las mismas. En nuestro caso la vamos a instalar en el mismo equipo que el director, en `servCopias.dtic.ua.es`, pero tal y como sucedía con el resto de componentes de Bacula, podríamos haberlo instalado en cualquier otro equipo. Para ello ejecutamos:

```
servCopias.dtic.ua.es# apt-get install bacula-console
```

En el fichero de configuración, /etc/bacula/bconsole, definiremos los datos del director al que nos conectamos.

```
Director {
  Name = servCopias-director
  DIRport = 9101
  address = servCopias.dtic.ua.es
  Password = "password"
}
```

Listado 5.15. Fichero de configuración, /etc/bacula/bconsole, del Bacula Console.

A continuación ya podremos entrar en la consola de Bacula, ejecutando:

```
servCopias.dtic.ua.es# bconsole
```

La consola nos ofrece una gran variedad de comandos para comprobar el estado las copias y los componentes de Bacula. Mediante el comando `status` podemos ver el estado del Director, de los Storage, de los Client o de todos. Si, por ejemplo, elegimos ver el estado del Director, nos mostrará los trabajos que tiene pendientes, en ejecución y terminados. El listado que nos muestra de los trabajos es bastante detallado, indicándonos el tipo de copia que se hizo, así como el número y tamaño de los archivos que la componen, y si se realizó correctamente, además de su nombre. También podemos ver en las tareas pendientes (Scheduled jobs) el próximo tipo de copia y cuando se realizará (según las hayamos definido en el Schedule correspondiente), así como el nombre que recibirá (según hayamos definido mediante el LabelFormat del pool).

```
*status
Status available for:
     1: Director
     2: Storage
     3: Client
     4: All
Select daemon type for status (1-4): 1
pruebas-director Version: 1.36.2 (28 February 2005) i386-pc-linux-
gnu debian 3.1
Daemon started 05-Sep-06 12:01, 2 Jobs run since started.

Scheduled Jobs:
Level  Type   Pri     Scheduled       Name           Volume
===================================================================
Diff  Backup  10  10-Sep-06 1:05  servWeb-mensual servWeb20061005
Diff  Backup  10  10-Sep-06 1:05  servBD-mensual  servWeb20061005
====

Running Jobs:
No Jobs running.
====

Terminated Jobs:
JobId  Level  Files Bytes  Status Finished          Name
===================================================================
1      Full   134    84,703  OK   08-Sep-06 1:06  servWeb-mensual
2      Full    81   342,625  OK   08-Sep-06 1:06  servBD-mensual
3      Incr     2     2,114  OK   09-Sep-06 1:06  servWeb-mensual
4      Incr     8    28,216  OK   09-Sep-06 1:06  servBD-mensual
====
*
```

Listado 5.16. Resultado de mostrar el estado del Director en la consola de Bacula.

Una vez terminados los trabajos, el director genera una serie de mensajes detallados indicándonos el resultado de los mismos, los cuales podemos ver invocando el comando `messages`.

Mediante la ejecución desde la consola del comando `help` podremos ver la lista de comandos de la consola de Bacula, los cuales nos permitirán un gran número de operaciones sobre las copias, tales como la ejecución

```
*restore

First you select one or more JobIds that contain files
to be restored. You will be presented several methods
of specifying the JobIds. Then you will be allowed to
select which files from those JobIds are to be restored.

To select the JobIds, you have the following choices:
     1: List last 20 Jobs run
     2: List Jobs where a given File is saved
     3: Enter list of comma separated JobIds to select
     4: Enter SQL list command
     5: Select the most recent backup for a client
     6: Select backup for a client before a specified time
     7: Enter a list of files to restore
     8: Enter a list of files to restore before a specified time
     9: Cancel
Select item:  (1-9): 8

The restored files will the most current backup
BEFORE the date you specify below.

Enter date as YYYY-MM-DD HH:MM:SS :2006-09-09 13:00:00
Select Client: servWeb-fd
Enter file names with paths, or < to enter a filename
containg a list of file names with paths, and terminate
them with a blank line.
Enter full filename: /var/www/index.html
Enter full filename:
Bootstrap records written to /var/lib/bacula/restore.bsr


The job will require the following Volumes:

   servWeb-20060909



1 file selected to be restored.
```

Listado 5.17. Primera parte de listado de la ejecución de la recuperación del archivo.

de trabajos, la creación y eliminación de pools, renombrado de las copias, … De estas operaciones, nos vamos a centrar, a modo de ejemplo, en la recuperación de una copia de seguridad, al ser la más útil de todas ellas tal y como se espera de un sistema de copia de seguridad. Como escenario de partida del estado de nuestro sistema de seguridad vamos a tomar el mostrado en el listado 5.16, y vamos a suponer que uno de los programadores de las páginas de nuestro servidor web ha borrado accidentalmente la página /var/www/index.html el día 9 de Septiembre de 2006 a las 2 de la tarde, por lo que necesitamos devolver dicha página al último estado anterior a ese del que tenemos copias de seguridad. Por lo tanto, deberemos recuperar dicha página a partir de la última copia de seguridad que se hizo antes del borrado, para lo cual ejecutaremos desde la consola el comando `restore`:

```
Run Restore job
JobName:    servWeb-restore
Bootstrap:  /var/lib/bacula/restore.bsr
Where:      *None*
Replace:    always
FileSet:    dirServWeb
Client:     servWeb-fd
Storage:    servCopias-sd
When:       2006-09-09 1:05:34
Catalog:    MyCatalog
Priority:   10
OK to run? (yes/mod/no): yes
Job started. JobId=9
*
```

Listado 5.18. Segunda parte de listado de la ejecución de la recuperación del archivo.

Mediante el comando `messages` comprobamos el resultado:

```
*messages
05-Sep 13:04 servCopias-director: Start Restore Job servWeb-resto-
re.2006-09-09_17.04.19
09-Sep 17:04 servCopias-sd: Ready to read from volume "servWeb-
20060909" on device /copias.
servWeb-fd: -rw-r--r--   1 root     root         1773 2006-09-08
12:01:34  /var/www/index.html
09-Sep 17:04 servCopias-director: Bacula 1.36.2 (28Feb05): 09-Sep-
2006 17:04:23
  JobId:                  9
  Job:                    servWeb-restore.2006-09-09_17.04.19
  Client:                 servWeb-fd
  Start time:             09-Sep-2006 17:04:21
  End time:               09-Sep-2006 17:04:23
  Files Expected:         1
  Files Restored:         1
  Bytes Restored:         1,773
  Rate:                   0.9 KB/s
  FD Errors:              0
  FD termination status:  OK
  SD termination status:  OK
  Termination:            Restore OK

05-Sep 13:04 pruebas-director: Begin pruning Jobs.
05-Sep 13:04 pruebas-director: No Jobs found to prune.
05-Sep 13:04 pruebas-director: Begin pruning Files.
05-Sep 13:04 pruebas-director: No Files found to prune.
05-Sep 13:04 pruebas-director: End auto prune.

*


*
```

Listado 5.19. Resultado de la recuperación del archivo /var/www/index.htm.

Como hemos visto en este ejemplo, mediante el Bacula Console disponemos de una gran variedad de herramientas para supervisar los trabajos realizados y realizar recuperaciones a partir de los mismos de una forma sencilla, facilitando así nuestra gestión de las copias de seguridad.

6. CORREO ELECTRÓNICO

El correo electrónico es uno de los servicios de red más utilizados en Internet desde sus inicios, siendo un elemento clave en cualquier organización. En este capítulo veremos las fases que atraviesa un mensaje desde que es enviado por un usuario hasta que lo recibe el destinatario, así como los agentes que intervienen y los distintos protocolos utilizados durante las distintas fases.

Como caso de uso mostraremos como instalar y configurar un servidor de correo para sistemas unix, basado en software libre, que proporcione servicio de correo electrónico a una organización.

6.1. ARQUITECTURA

A la hora de enviar o recibir un mensaje de correo, el usuario solo ve el programa de correo electrónico que utiliza, cuando en realidad entran en funcionamiento otras aplicaciones de correo electrónico que realizan una determinada función en el proceso de mover y manipular los mensajes. En general, dichas aplicaciones pueden dividirse en una de las siguientes categorías:

Agente de usuario de correo: El MUA (Mail User Agent) es un programa que permite al usuario leer y crear mensajes de correo electrónico, cayendo en esta categoría la mayoría de los clientes de correo electrónico (Microsoft Outlook, Mozilla Thunderbird, Webmail, …). La mayoría de los MUA actuales también permiten a los usuarios obtener los mensajes y configurar sus buzones.

Agente de transferencia de correo: El MTA (Mail Transfer Agent) transfiere los mensajes de correo entre equipos mediante el protocolo SMTP. Un mensaje puede pasar por varios MTA hasta llegar a su destino. Los MTA son capaces de transferir correo local (en la misma máquina) y remoto (reenviarlo a otro MTA para que lo procese). Para evitar problemas de spam, el MTA debe estar correctamente configurado para prevenir dichos ataques.

Agente de entrega de correo: El MTA utiliza un MDA (Mail Delivery Agent) para entregar los mensajes en el buzón del usuario correspondiente. En muchos casos el MDA es en realidad un agente de entrega local (Local Delivery Agent o LDA). Los MDA no transportan mensajes entre sistemas ni se relacionan con el usuario final, y una ventaja que ofrecen es la de poder ser utilizados para la clasificación de los mensajes antes de que el usuario los lea.

A continuación vamos a ver como intervienen estos agentes cuando un usuario envía un mensaje (Figura 6.1). Para clarificar el ejemplo, vamos a suponer que el usuario `usuario1` del dominio `mired.dtic.ua.es` envía un mensaje al usuario `usuario2` del `dominio otrared.dtic.ua.es`:

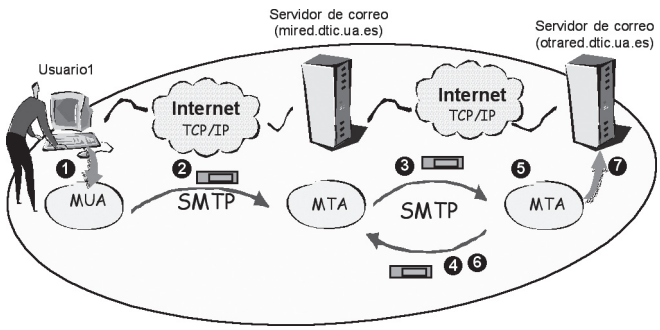


Figura 6.1. Pasos del mensaje durante el envío.

❶ En primer lugar el usuario con la cuenta de correo `usuario1@mired.dtic.ua.es` redacta su mensaje con su programa de correo electrónico (su MUA), con destinatario `usuario2@otrared.dtic.ua.es`, y le ordena que lo envíe.

❷ Una vez que ha recibido la orden de enviar el mensaje, el MUA puede añadir cabeceras adicionales al mensaje, tales como la fecha o sustitución de direcciones, para proceder a inyectar el mensaje en el sistema de correo. Para realizar dicha inyección del mensaje el MUA puede utilizar un programa o hablar directamente el protocolo SMTP con el MTA del sistema local o del sistema remoto de correo (el servidor de correo de `mired.dtic.ua.es`).

❸ Si la inyección ha tenido éxito, el mensaje ya es responsabilidad del MTA (el MTA de `mired.dtic.ua.es`), que examina el encabezado para determinar a donde enviar el mensaje. Si la cuenta del destinatario hubiera pertenecido al MTA de `mired.dtic.ua.es`, el mensaje se habría entregado localmente, pero en este caso la cuenta esta en otro servidor, por lo que el MTA de `mired.dtic.ua.es` establece una conexión SMTP con el MTA de `otrared.dtic.ua.es` y le pasa el mensaje. Dicho mensaje se envía en dos partes: el sobre, que especifica la dirección del destinatario (`usuario2@otrared.dtic.ua.es`) y la de retorno (`usuario1@mired.dtic.ua.es`); y el propio mensaje, compuesto por la cabecera y el cuerpo.

❹ Si el MTA de `otrared.dtic.ua.es` rechaza el mensaje (no exista el usuario `usuario2`, no se permitan mensajes desde nuestra red, …) entonces el MTA de la maquina `mired.dtic.ua.es` enviará un mensaje de devolución a la dirección de retorno.

❺ Si el MTA de `otrared.dtic.ua.es` acepta el mensaje, entonces examina la dirección del destinatario, y al ver que es una cuenta local lo entrega el mismo en el buzón del usuario `usuario2` o delega dicha entrega en un MDA. Si la cuenta hubiera sido de un equipo remoto habría procedido del mismo modo que hizo el MTA de `mired.dtic.ua.es` en el paso 3, abriendo una conexión SMTP con el sistema remoto en cuestión.

❻ Si no se puede realizar la entrega (el `usuario2` tiene bloqueada su cuenta, o ha excedido el tamaño de su buzón, …), entonces el MTA de (`otrared.dtic.ua.es`) envía un mensaje de devolución a la dirección de retorno que aparece en el sobre (`usuario1@mired.dtic.ua.es`).

❼ Si se ha conseguido entregar el mensaje, se almacenará en el buzón del usuario `usuario2` hasta que su MUA lo lea.

6.2. ENVÍO DE MENSAJES MEDIANTE SMTP

El Protocolo Sencillo de Transferencia de Correo (Simple Mail Transfer Protocol o SMTP) es el protocolo utilizado para el envío de mensajes

de correo desde el MUA del cliente al MTA del servidor de correo, y el utilizado en las comunicaciones entre los MTA's de los servidores de correo.

SMTP (definido en la RFC-821) utiliza el puerto 25 del servidor para la comunicación, y el 465 para el modo seguro (SSMTP, sobre SSL). Es un protocolo que no requiere autenticación, lo que permite a cualquier persona conectarse al sistema para enviar correos, con el consiguiente problema del envío de correo basura (spam). Afortunadamente, los servidores SMTP modernos (MTA) proveen mecanismos que minimizan dicho problema, limitando el reenvío y permitiendo que sólo hosts conocidos envíen mensajes.

Posteriormente se han añadido diversas extensiones al protocolo básico, tales como: identificación de las extensiones que soporta el servidor (RFC-1869), autenticación (RFC-2554) y uso de códigos de errores estándar (RFC-2034).

6.3. RECUPERACIÓN DE MENSAJES

A la hora de consultar el correo mediante nuestro MUA, tenemos la posibilidad de escoger entre dos protocolos: POP e IMAP.

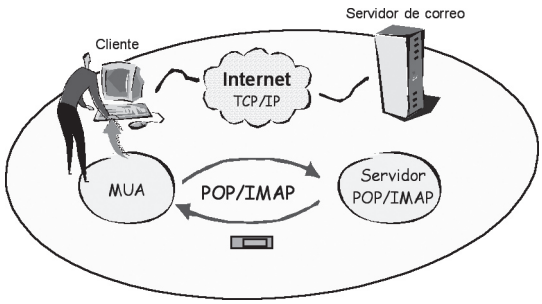


Figura 6.2. Recuperación de mensajes.

6.3.1. Protocolo POP3

El protocolo de oficina de correo (Post Office Protocol o POP) permite a los clientes de correo electrónico descargar sus mensajes desde servidores remotos y guardarlos en su máquina local. Una vez descargados, permite borrar dichos mensajes del servidor o no, según hayamos configurado nuestro cliente de correo.

POP3 (definido en la RFC-1939) es la versión actual del protocolo, y utiliza el puerto 110 del servidor para la comunicación, y el 995 para el modo seguro (POP3S, sobre SSL). Al establecer la conexión, el servidor POP envía al cliente un saludo, y comienza un intercambio de mensajes, entre los cuales el servidor solicita al cliente que se identifique (estado de autenticación) y el cliente le envía su nombre de usuario y contraseña. Si la autenticación tiene éxito, el cliente POP pasa al estado de transacción en el que podrá utilizar comandos para enumerar, descargar y eliminar los mensajes del servidor. Los mensajes a eliminar no se borran hasta que el cliente finaliza la sesión, momento en el que el servidor POP entrará en estado de actualización y borrará los mensajes y los recursos utilizados durante la sesión.

También se dispone de variantes, menos utilizadas, del protocolo POP:

APOP: POP3 con autenticación MD5 de la contraseña.
KPOP: POP3 con autenticación Kerberos. Utiliza el puerto 1109.

6.3.2. Protocolo IMAP

El Protocolo de Acceso a Mensajes de Internet (Internet Message Access Protocol o IMAP) permite a los clientes de correo acceder a mensajes guardados remotamente. Los mensajes permanecen en el servidor de correo remoto, donde los usuarios pueden leerlos o eliminarlos, así como crear, renombrar o eliminar buzones de correo. Es compatible con las Extensiones de Correo de Internet Multipropósito (MIME) y utiliza menor ancho de banda que POP3 al extraer inicialmente sólo la información de la cabecera del mensaje, permitiendo al usuario ver el tamaño de los archivos adjuntos sin necesidad de descargar el mensaje. Además, permite eliminar mensajes sin tener que ver el cuerpo de los mismos.

IMAP (definido en la RFC-1730), y la versión actual del protocolo (IMAP4, RFC-2060), utilizan el puerto 143 del servidor para la comunicación, y el 993 para el modo seguro (IMAPS, sobre SSL).

A la hora de consultar el correo, la filosofía utilizada por POP es más adecuada para usuarios que utilizan un solo equipo para dichas consultas, al descargar todos los mensajes en la misma máquina, mientras que IMAP es más adecuado para usuarios que utilizan varias máquinas para consultar el correo y/o con conexiones de poco ancho de banda.

6.4. GESTORES DE CORREO

Los gestores de correo actuales pueden considerarse como las versiones modernas de los MUA tradicionales, no limitándose a solo enviar y recibir mensajes, sino que ofrecen otras muchas funcionalidades, tales como mantenimiento de libretas de direcciones, calendarios, sindicación por RSS, subscripción a grupos de noticias, etc…

En la actualidad existen un gran número de gestores de correo, tanto comerciales como gratuitos, siendo los más utilizados, con gran diferencia sobre el resto, Microsof Outlook y Mozilla Thunderbird. En este apartado vamos a ver como configurar correctamente dichos programas para el envío por SMTP y recepción por POP3 o IMAP de nuestros mensajes de correo. En ambos casos vamos a suponer que disponemos de la cuenta de correo usuario1 en el servidor de correo correo.dtic.ua.es, y que en dicho servidor se encuentran en ejecución servidores SMTP, POP3 e IMAP, todos ellos sobre SSL para asegurar la transmisión.

En primer lugar vamos a configurar Microsoft Outlook , para lo cual debemos crear una nueva cuenta mediante las opciones de menú: `Herramientas->Cuentas de correo electrónico`, eligiendo a continua-

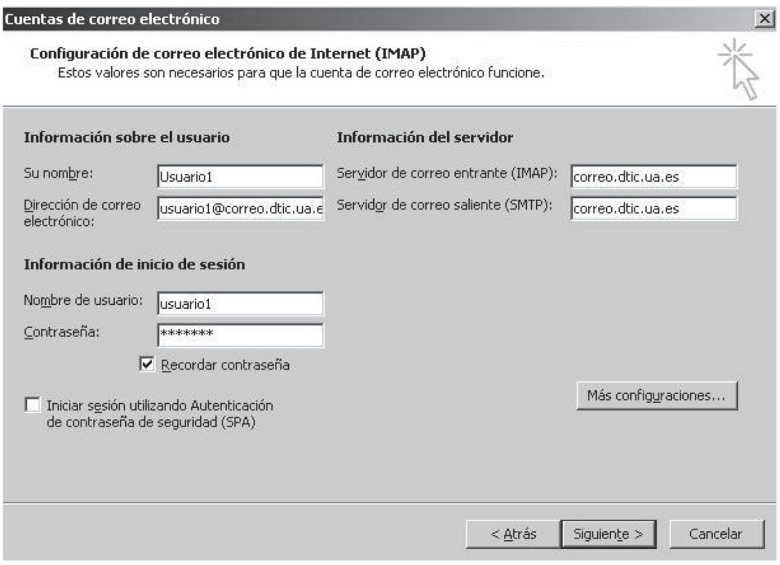


Figura 6.3. Configuración de la cuenta de correo en Microsoft Outlook.

ción agregar una nueva cuenta, seleccionando como tipo de servidor (POP o IMAP) el que prefiramos para recibir nuestros mensajes, y rellenando finalmente el formulario con los datos de nuestra cuenta y servidor.

En el caso de querer que la comunicación vaya por una conexión segura, mediante SSL, deberemos seleccionar el botón de `Más configuraciones`, y seleccionar en la ventana emergente que nos aparece la opción `Avanzadas`.

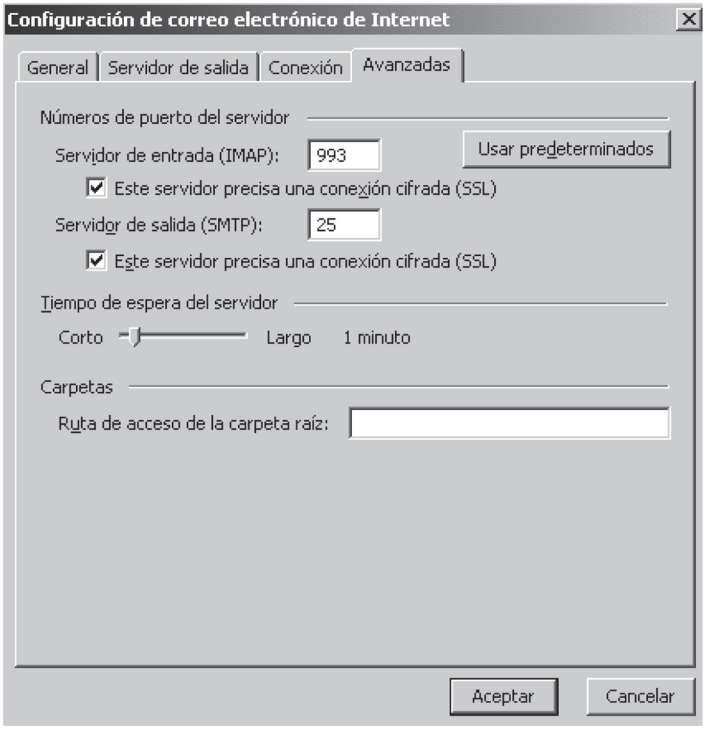


Figura 6.4. Configuración de Microsoft Outlook para realizar la comunicación por SSL.

En caso de haber seleccionado en pasos anteriores la recepción de mensajes por POP3, en lugar de aparecernos la opción de IMAP sobre SSL y puerto 993 nos hubiera aparecido POP3 sobre SSL y puerto 995.

De este modo ya podremos enviar mensajes por SMTP y recibir por POP3 o IMAP según hayamos elegido, y mediante una conexión segura.

A continuación vamos a configurar la misma cuenta en Mozilla Thunderbird. Comenzamos seleccionando `Fichero->Nueva Cuenta` en el menú, y de tipo Cuenta de correo. En la siguiente pantalla introduciremos nuestro nombre, `usuario1`, y nuestra cuenta, `usuario1@dtic.ua.es`. Seleccionaremos el método de recepción de mensajes entre POP3 e IMAP y como servidor de correo entrante indicaremos correo.dtic.ua.es. Por último, como nombre del usuario de recepción de correo pondremos `usuario1`, y como cuenta `usuario1@correo.dtic.ua.es`, obteniendo como resultado:

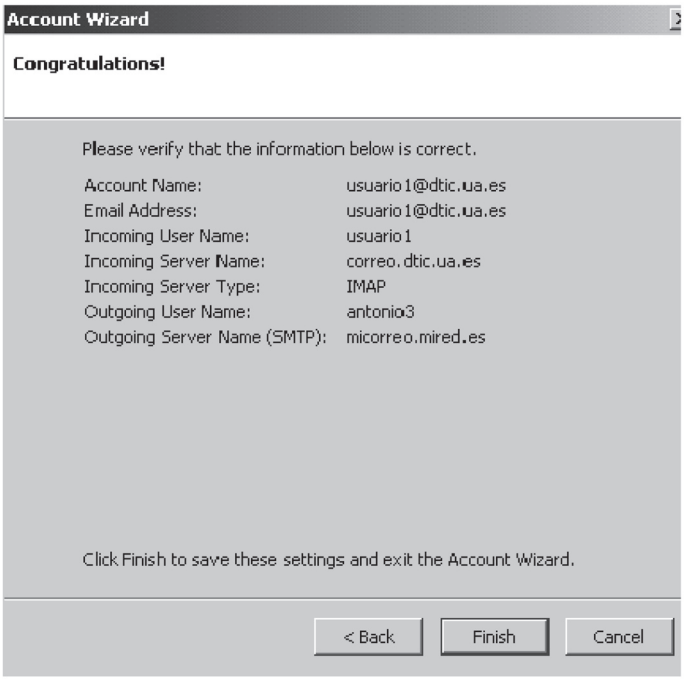


Figura 6.5. Configuración de la cuenta de correo en Mozilla Thunderbird.

Para realizar la comunicación sobre SSL deberemos pulsar con el botón derecho del ratón sobre la cuenta recién creada, y seleccionar `Propiedades`, y en la ventana que aparece seleccionar el submenú Configuración del Servidor, para, por último, seleccionar la opción que habilita la comunicación por SSL.

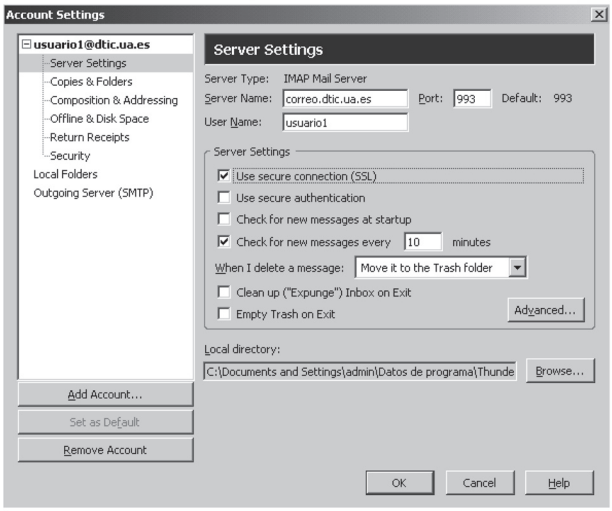


Figura 6.6. Configuración de Mozilla Thunderbird para realizar la comunicación por SSL.

6.5. PASARELAS WEB

Las pasarelas Webmail son aplicaciones web que nos permiten acceder al correo electrónico utilizando como MUA un navegador web, y por lo tanto, usando como protocolo de comunicación el HTTP o HTTPS (puertos 80 y 443 respectivamente) para el envío y recepción de los mensajes, en lugar de los protocolos SMTP, POP3 e IMAP. Serán, en la mayoría de los casos, dichas pasarelas las que utilicen dichos protocolos para comunicarse con los servidores de correo, de forma que actúan de interfaz entre el usuario y dichos servidores. Entre los ejemplos más conocidos de webmail tenemos Hotmail y Gmail, de las empresas Microsoft y Google respectivamente, que ofrecen cuentas gratuitas a los usuarios.

Las principales ventajas que nos proporcionan son:

- Guardan la configuración en el servidor, de forma que podremos conectarnos con un navegador desde cualquier equipo sin tener que reconfigurar las propiedades de nuestro buzón (carpetas, filtros, apariencia, etc…).
- Al utilizar como MUA un navegador web, nos evita tener que instalar y/o utilizar un MUA específico.

- Nos permiten acceder al correo sin tener que preocuparnos de los firewalls que haya entre el servidor y el equipo desde el que intentamos conectarnos. Esto es debido a que muchos firewalls están configurados para solo permitir tráfico por los puertos 80 y 443 (HTTP y HTTPS), los cuáles no permitirían acceder con los protocolos SMTP, POP e IMAP que utilizan los MUA tradicionales.

Entre las aplicaciones webmail de código libre más difundidas tenemos openwebmail o squirrelmail. En el último apartado de este capítulo se explicará como instalar y configurar Squirrelmail.

6.6. CASO DE USO: QMAIL + COURIER-IMAPD + SQUIRRELMAIL

En este apartado vamos a configurar un servidor de correo que acepte mensajes mediante SMTP y permita la consulta de los mismos por POP3 e IMAP. Para ello utilizaremos qmail como servidor de SMTP y POP3, y courier-imap para IMAP. Nuestro sistema también será capaz de enviar y recibir correo por webmail mediante Squirrelmail.

6.6.1. Qmail

Qmail es un Agente de Transporte de Correo (MTA) para sistemas UNIX, desarrollado por Dan Bernstein (profesor de la Universidad de Illinois), que sustituye por completo a sendmail, el MTA tradicional de los sistemas UNIX. Es de libre distribución, pero no es licencia GPL, en concreto se puede redistribuir libremente distribuciones de su código fuente, pero sin modificaciones.

Entre las principales ventajas que proporciona están:

- Esquema modular: Cada etapa es llevada a cabo por un proceso distinto, en lugar de un único proceso como hacen otros MTA, permitiendo introducir fácilmente filtros entre las distintas etapas del procesamiento de un mensaje.
- Mayor seguridad: Sendmail y otros MTA se ejecutan como un único proceso con privilegios de administrador, lo cual puede provocar problemas de seguridad, mientras que en qmail la mayoría de los procesos que intervienen no requieren ser ejecutados como administrador.
- Permite la conversión fácil a partir de archivos de sendmail (tanto mensajes como listas de correo), y compatibilidad con dicho MTA,

al implementar sus propios ejecutables sendmail, de modo que el resto de programas del sistema que utilicen sendmail no se verán afectados por el cambio a qmail.

- Envío de mensajes concurrente (maneja hasta 20 colas de mensajes simultáneamente).

6.6.1.1. Arquitectura de qmail

La arquitectura de qmail sigue un diseño modular, encargándose cada proceso de una tarea específica, tal como se aprecia en la figura 6.7.

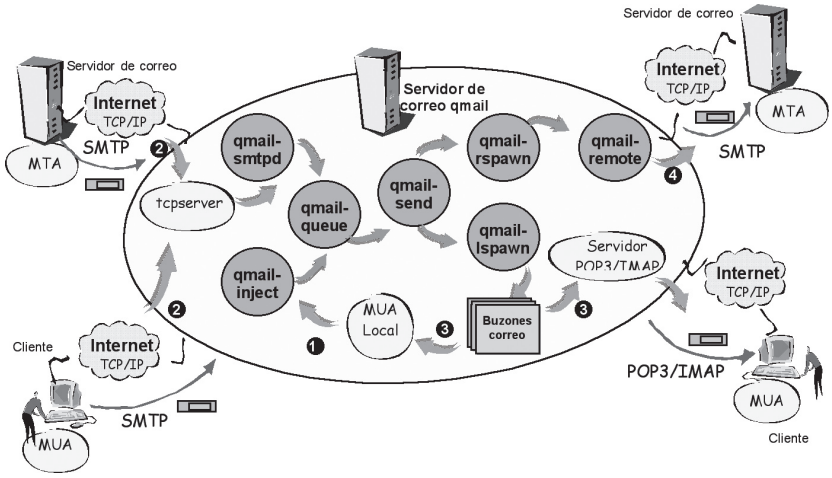


Figura 6.7. Arquitectura de Qmail.

Como se ve en la figura, según el tipo de origen y destino (local o remoto), el procesamiento del mensaje lo realizará un determinado proceso del sistema qmail:

1. Mensaje con origen local: El remitente envía un correo usando un MUA local (por ejemplo, el programa mail) desde la máquina del servidor de correo. En este caso qmail-inject enviará el mensaje al programa de encolamiento de correo, qmail-queue, el cual lo almacenará en la cola de mensajes /var/qmail/queue. A continuación qmail-send intentará enviarlo a su destino mediante qmail-lspawn o qmail-rspawn, según sea el destinatario local o remoto.

2. Mensaje con origen remoto: Al llegar un mensaje SMTP desde el exterior, ya sea desde un MUA de un equipo de nuestro dominio (el remitente pertenece a nuestra organización) o desde otro servidor de correo (si el destinatario del mensaje pertenece a nuestra organización), qmail-smtpd será el encargado de enviar el correo al programa de encolamiento, qumail-queue, y el resto del proceso será idéntico al caso anterior.

3. Destino local: Los mensajes con destinatario local son procesados por qmail-lspawn y almacenados por qmail-local en el buzón correspondiente del destinatario, pudiendo ser consultados desde la maquina local con los MUA locales (por ejemplo el programa mail), o desde otros equipos mediante los protocolos POP e IMAP.

4. Destino remoto: En este caso se enviará, mediante el programa qmail-rspawn, el mensaje por SMTP a un servidor remoto de SMTP (al servidor de correo de la organización a la que pertenece el destinatario).

6.6.1.2. Instalación

A la hora de instalar qmail, en primer lugar debemos desinstalar los paquetes de cualquier MTA (sendmail, exim, …) que venga preinstalado en el sistema, pues de otra forma qmail lo detectará y no permitirá la instalación. Además, para guardar la compatibilidad con el resto de programas que utilizan sendmail como MTA, qmail genera sus propios ejecutables sendmail.

Hay que indicar que en la configuración que vamos a utilizar de ejemplo, el acceso a las cuentas de correo se basará en la configuración que tengamos para validar usuarios para acceder a la máquina que haga de servidor (usuarios locales, ldap, nis, kerberos, …).

En primer lugar hay que instalar los paquetes de fuentes ucspi-tcp-src (un sustituto de inetd) y qmail-src. Una vez instalados, se procede a construir los paquetes con los ejecutables y a su instalación con los comandos build-ucspi-tcp y build-qmail. En un sistema Debian llamado micorreo.mired.dtic.ua.es los comandos que hay que ejecutar, si utilizamos apt-get, son:

```
micorreo.mired.dtic.ua.es# apt-get install ucspi-tcp-src
micorreo.mired.dtic.ua.es# build-ucspi-tcp
micorreo.mired.dtic.ua.es# apt-get install qmail-src
micorreo.mired.dtic.ua.es# build-qmail
```

Una vez instalado qmail, se habrán generado los usuarios necesarios para su correcto funcionamiento: alias, qmaild, qmails, qmailr, qmailq, qmaill y qmailp.

Tras estos pasos ya tenemos instalado qmail, pero habrá que realizar un par de ajustes de configuración antes de poder ejecutar el servicio, tal y como se explicará en el siguiente apartado.

6.6.1.3. Administración

Los archivos de configuración se encuentran en el directorio /var/qmail/control, que en las versiones modernas de qmail es un enlace simbólico al directorio /etc/qmail. Inicialmente, en este directorio solo disponemos del archivo rcpthosts y el directorio users, pero hay una gran cantidad de archivos de configuración que podremos crear y modificar dentro del directorio /etc/qmail para ajustar nuestro servicio de correo.

6.6.1.4. Configuración del dominio

En primer lugar debemos indicar el nombre del sistema en el archivo /etc/qmail/me, y el nombre del dominio en el archivo /etc/qmail/defaultdomain si no lo hemos especificado en el nombre del sistema. En nuestro caso de ejemplo, si nuestro dominio es `mired.dtic.ua.es` y nuestro servidor de correo se llama `correo`, los configuramos sobrescribiendo dichos archivos con los siguientes valores:

```
micorreo.mired.dtic.ua.es# echo "correo.mired.dtic.ua.es" >
/etc/qmail/me
micorreo.mired.dtic.ua.es# echo "mired.dtic.ua.es" > /etc/
qmail/defaultdomain
```

Qmail viene configurado por defecto para admitir solamente correo local, por lo que debemos especificar las máquinas o dominios de los que aceptamos correo, añadiéndolos, uno por línea, al archivo /etc/qmail/rcpthosts. En nuestro caso añadimos las líneas necesarias para aceptar correo local y de los equipos de nuestro dominio, quedando el archivo:

```
localhost
micorreo.mired.dtic.ua.es
mired.dtic.ua.es
```

Listado 6.1. Contenido del archivo /etc/qmail/rcphosts.

También debemos añadir en el archivo /etc/qmail/locals las máquinas y dominios para las cuales debemos entregar el correo localmente en el servidor de correo. En nuestro caso:

```
localhost
micorreo.mired.dtic.ua.es
mired.dtic.ua.es
```

Listado 6.2. Contenido del archivo /etc/qmail/locals.

Si no hiciéramos esto, deberíamos especificar la dirección completa del servidor de correo en las cuentas de los usuarios, por lo que dichas cuentas deberían ser de la forma `usuario1@micorreo.mired.dtic.ua.es`. De este modo dichas cuentas también podrán ser del tipo `usuario1@mired.dtic.ua.es`, lo cual es más usual.

6.6.1.5. Elección del modo de almacenamiento

A la hora de configurar el modo de almacenamiento de los mensajes en el servidor qmail ofrece varios modos, para lo cual habrá que inicializar correctamente la variable `alias_empty` del archivo /etc/init.d/qmail. La configuración especificada mediante dicha dicha variable afectará a todos los usuarios, excepto a los que tengan un método personalizado, el cual se definiría en el archivo `.qmail` del directorio $HOME del usuario. Los métodos que nos ofrece qmail son:

- Entrega a un programa: De este modo se redirecciona el mensaje para que sea procesado por otro programa en lugar de depositarlo directamente en el buzón del destinatario. En el archivo /etc/init.d/qmail aparece la línea que haría que los mensajes recibidos fueran reenviados a procmail para su procesamiento:

```
alias_empty = "| /usr/sbin/qmail-procmail"
```

por lo que la línea que aparecería en un archivo `.qmail` de un usuario para obtener el mismo comportamiento sería:

```
| /usr/sbin/qmail-procmail
```

- Mbox: este es el modo utilizado por sendmail y el estándar de unix, en el cual se almacenan todos los mensajes en un único archivo llamado Mailbox en el directorio $HOME del usuario.

alias_empty = "./Mailbox"

- Maildir: surgió para evitar las limitaciones del formato mbox. En este formato los mensajes se almacenan en el directorio $HOME del usuario, dentro del subdirectorio Maildir, el cual a su vez contiene 3 subdirectorios: new, cur y tmp. Cada mensaje se guarda en un archivo separado, el cual se alojará en uno de los tres subdirectorios anteriores: en new si es un correo sin leer, en cur si es un mensaje ya leído, y en tmp si se esta entregando. Se configuraría mediante:

```
alias_empty = "./Maildir"
```

A la hora de utilizar esta opción, hay que tener en cuenta que el directorio Maildir y sus subdirectorios deben estar ya creados para que funcione el almacenamiento de los correos recibidos, por lo cuál habrá que generar dicha estructura de directorios dentro del $HOME del usuario. Para ello contamos con la herramienta `maildirmake`, que nos genera dicha estructura entrando en cada directorio $HOME de cada usuario ya creado y ejecutando

```
maildirmake Maildir
```

y asignándole el usuario y grupo adecuados al directorio recién creado.

Para que se cree automáticamente dicha estructura de directorios para los nuevos usuarios que creemos, debemos ejecutar la orden anterior dentro del directorio /etc/skel.

6.6.1.6. Configuración del reenvío de mensajes

El relaying (reenvío de correos) sucede cuando un MTA acepta un mensaje SMTP cuyo destinatario o remitente no es una cuenta local. En qmail inicialmente se encuentra desactivado, por lo cuál debemos especificar a que equipos o redes permitimos el reenvío de correo. Para ello deberemos añadir en el archivo /etc/tcp.smtp líneas de este tipo indicando el nombre o dirección del equipo o subred al que permitimos el reenvío,

por lo que en nuestro caso, para permitir solo el reenvío para equipos de nuestro dominio (mired.dtic.ua.es) lo configuraremos con los valores:

```
mired.dtic.ua.es:allow,RELAYCLIENT=""
```

Listado 6.3. Contenido del archivo /etc/tcp.smtp.

y deberemos ejecutar, dentro del directorio /etc, la siguiente orden para que los cambios tengan efecto:

```
micorreo.mired.dtic.ua.es# tcprules tcp.smtp.cdb tcp.smtp.temp
< tcp.smtp
```

De este modo, el demonio tcpserver le dirá a qmail-smtpd que ignore el archivo rcpthosts para dichas direcciones.

6.6.1.7. Alias

Los archivos de alias en qmail se almacenan en el directorio /var/qmail/alias. Estos archivos son de tipo oculto, siguiendo la siguiente nomenclatura `.qmail-nombreUsuario`, y en dichos archivos se debe especificar a que cuentas se redirigen los correos con dicho destinatario. De este modo, si, por ejemplo, quisiéramos que los mensajes que tienen como destinatario la dirección de correo `grupo1@mired.dtic.ua.es` le llegaran a las cuentas de los usuarios `usuario1` y `usuario2`, deberíamos crear el siguiente archivo /var/qmail/alias/.qmail-grupo1:

```
usuario1
usuario2
```

Listado 6.4. Contenido del archivo /var/qmail/alias/.qmail-grupo1.

También debemos definir un alias para redirigir el correo que tenga como destinatario el usuario root a la cuenta de otro usuario, ya que por motivos de seguridad dicho usuario no puede recibir correos. Por lo tanto, será necesario especificar en el archivo .qmail-root la(s) cuenta(s) a donde se enviarán realmente dichos mensajes.

En este momento ya tenemos configurado nuestro servidor para recibir correos y almacenarlos en los buzones de los usuarios correspondientes. En los siguientes apartados veremos como instalar los servidores necesarios para que los clientes puedan recuperar dichos mensajes mediante sus MUA utilizando los protocolos POP3 e IMAP.

6.6.1.8. Instalación del servidor POP3

Qmail viene con su servidor de POP3, qmail-pop3d, el cuál no está activado por defecto. Para ello debemos editar el script de arranque de qmail, /etc/init.d/qmail, y descomentar las siguientes líneas de las secciones start y stop:

```
# Uncomment the following lines to start the pop3 server
sh -c "start-stop-daemon --start --quiet --user root \
    --pidfile /var/run/tcpserver_pop3d.pid --make-pidfile \
    --exec /usr/bin/tcpserver -- -R -H \
    0 pop-3 /usr/sbin/qmail-popup `hostname`.`dnsdomainname`\
    /usr/bin/checkpassword /usr/sbin/qmail-pop3d Maildir &"
```

Listado 6.5. Líneas de la sección start del archivo /etc/init.d/qmail necesarias para activar el servidor qmail-pop3d.

```
#Uncomment the next line if you have enabled the pop3 server
start-stop-daemon --user root --stop --quiet --oknodo \
      --pidfile /var/run/tcpserver_pop3d.pid --exec  \ /usr/bin/
      tcpserver
```

Listado 6.6. Líneas de la sección stop del archivo /etc/init.d/qmail necesarias para activar el servidor qmail-pop3d.

A continuación iniciamos la ejecución de qmail, el cuál ya podra aceptar mensajes mediante SMTP y consultas de los mismos por parte de los clientes mediante POP3, con el comando:

```
micorreo.mired.dtic.ua.es# /etc/init.d/qmail start
```

6.6.2. Instalación del servidor IMAP

Como servidor de IMAP vamos a utilizar el programa courier-imap. Para instalarlo basta con ejecutar el comando:

```
micorreo.mired.dtic.ua.es# apt-get install courier-imap
```

Los archivos de configuración de dicho servidor se encuentran en el directorio /etc/courier:

- En el archivo imapd podemos configurar el comportamiento general del servidor, pudiendo definir la dirección y número de puerto (por defecto el 143) que escuchará, el número máximo de conexiones simultáneas permitidas, etc… También indicamos aquí el método de autenticación de los clientes (por defecto delega la configuración en el archivo authdaemonrc) y de almacenamiento de mensajes (Maildir, Mailbox, …)
- Mediante el archivo authdaemonrc se configura el método de autentificación que utilizarán los usuarios para acceder a sus buzones, el cuál por defecto viene configurado (authmodulelist="authpam") para utilizar el método que usemos para acceder al sistema (usuarios locales, ldap, kerberos, …).

Si además queremos instalar el servidor con encriptación SSL en la comunicación con los clientes, instalamos el paquete courier-imap-ssl:

```
micorreo.mired.dtic.ua.es# apt-get install courier-imap-ssl
```

La configuración de dicho servidor se realiza mediante el archivo imapd-ssl, que se encuentra en el mismo directorio que los anteriores, y es muy similar al imapd.

De este modo dispondremos de un servidor de IMAP escuchando el puerto 143 y de otro que escucha IMAP seguro en el puerto 993, pudiendo elegir el cliente en la configuración de su MUA el protocolo de comunicación que utilizará para consultar los mensajes.

6.6.3. Instalación y configuración de Squirrelmail

Squirrelmail es una pasarela webmail de código libre, que funciona sobre el servidor web Apache y escrita en PHP, que nos permite consultar

nuestro correo mediante IMAP. Una de sus principales ventajas es que dispone de una gran cantidad de pluggins que podemos descargar de la página web www.squirrelmail.org para ampliar sus funcionalidades, tales como filtros antispam, revisión ortográfica, etc,...

En nuestro caso vamos a instalarlo en el mismo sistema en el que hemos instalado los servidores de correo, mediante el comando:

```
micorreo.mired.dtic.ua.es# apt-get install squirrelmail
```

A continuación ejecutamos el script de configuración:

```
micorreo.mired.dtic.ua.es# /etc/squirrelmail/conf.pl
```

La configuración de squirrelmail es bastante intuitiva gracias a dicho script, apareciéndonos el menú principal de configuración:

```
SquirrelMail Configuration : Read: config.php (1.4.0)
---------------------------------------------------------
Main Menu --
1.  Organization Preferences
2.  Server Settings
3.  Folder Defaults
4.  General Options
5.  Themes
6.  Address Books
7.  Message of the Day (MOTD)
8.  Plugins
9.  Database
10. Languages

D.  Set pre-defined settings for specific IMAP servers

C   Turn color on
S   Save data
Q   Quit

Command >>
```

Listado 6.7. Menú principal de configuración de Squirrelmail.

El primer submenú nos permitirá definir la apariencia de la pantalla de acceso a webmail. Con el segundo podremos configurar el servidor IMAP y de SMTP que utilizaremos para consultar y enviar mensajes.

```
SquirrelMail Configuration : Read: config.php (1.4.0)
---------------------------------------------------------
Server Settings

General
-------
1.  Domain                 : mired.dtic.ua.es
2.  Invert Time            : false
3.  Sendmail or SMTP       : SMTP

SMTP Settings
-------------
4.   SMTP Server           : localhost
5.   SMTP Port             : 25
6.   POP before SMTP       : false
7.   SMTP Authentication   : none
8.   Secure SMTP (TLS)     : false

A.  Update IMAP Settings   : localhost:143 (other)
H.  Hide SMTP Settings

R   Return to Main Menu
C   Turn color on
S   Save data
Q   Quit

Command >>
```

Listado 6.8. Submenú de configuración de los servidores a utilizar por Squirrelmail.

Con la opción 8 del menú principal podremos añadir nuevos pluggins, y con la opción 10 el lenguaje a utilizar, en nuestro caso el castellano, para lo que deberemos seleccionar las siguientes opciones:

```
SquirrelMail Configuration : Read: config.php (1.4.0)
---------------------------------------------------------
Language preferences
1.  Default Language       : es_ES
2.  Default Charset        : es_ES.ISO-8859-1
3.  Enable lossy encoding  : false

R   Return to Main Menu
C   Turn color on
S   Save data
Q   Quit

Command >>
```

Listado 6.9. Configuración de squirrelmail para utilizar el castellano.

Además, si no tenemos habilitado ese juego de caracteres (es_ES.ISO-8859-1) en el sistema, al acceder a nuestro webmail nos aparecerán las opciones en inglés, por lo que deberemos habilitarlo ejecutando:

```
micorreo.mired.dtic.ua.es# dpkg-reconfigure locales
```

Por último, para que Apache pueda servir la aplicación de webmail deberemos crear el siguiente enlace simbólico:

```
micorreo.mired.dtic.ua.es# ln -s /usr/share/squirrelmail /
var/www/webmail
```

Una vez reiniciado el servidor web Apache ya podremos acceder a la pantalla de acceso de nuestro webmail mediante la dirección http://micorreo.mired.dtic.ua.es/webmail:

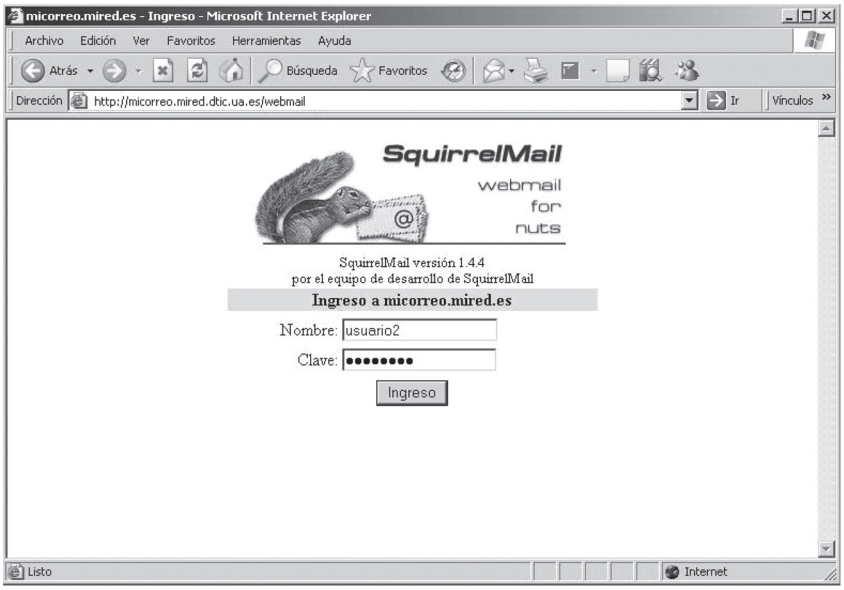


Figura 6.8. Pantalla de ingreso a nuestro webmail.

Debemos tener en cuenta que la validación de los usuarios la hace en realidad contra el servidor IMAP que utilicemos, y por tanto utiliza-

rá el método definido en dicho servidor, que en nuestro caso (el servidor courier-imap) utilizaba usuarios locales. Además, hay que indicar que si queremos que la comunicación sea segura, utilizando HTTPS (puerto 443), solo tendremos que configurar el servidor Apache para que sirva dicho dominio mediante SSL.

Una vez validado nuestro usuario podremos crear y consultar nuestros mensajes de correo, además de una gran cantidad de opciones de configuración de nuestro buzón, tales como filtrado de mensajes, configuración de las preferencias de visualización, etc… las cuales, además, podremos ir ampliando con la instalación de nuevos pluggins.

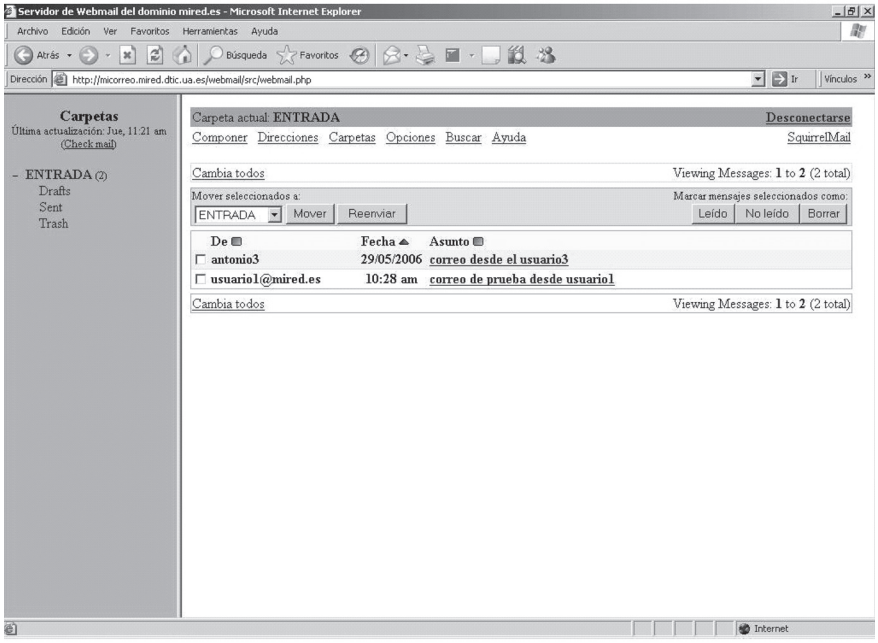


Figura 6.9. Pantalla mediante la que manipulamos nuestro buzón de correo mediante Squirrelmail.

PARTE II
SERVICIOS AVANZADOS PARA LA RED

7. SEGURIDAD

La gestión de la seguridad informática y de comunicaciones en una organización es un aspecto muy complejo, tanto que existen normas internacionales ISO desarrolladas para ayudar a implantar una buena seguridad integral en los sistemas informáticos de las organizaciones. La norma ISO 17799 para definir las políticas de seguridad es un buen ejemplo.

La seguridad de las redes informáticas y las comunicaciones se debe abordar desde cinco aspectos o perspectivas distintas:

- Prevención
- Detección
- Recuperación
- Análisis forense
- Evaluación

Siendo todos los enfoques importantes, se debe prestar especial atención al aspecto de la prevención; establecimiento de políticas de seguridad, definición de listas de control de acceso y diseño e implantación de mecanismos de seguridad perimetral.

Este tema se dedica a uno de los mecanismos de prevención, el establecimiento de comunicaciones seguras basadas fundamentalmente en el cifrado de la información y la utilización de protocolos de comunicación que hagan uso de este cifrado.

7.1. CIFRADO

La criptografía ("escritura secreta") es una rama de las matemáticas que se encarga del cifrado de los mensajes para que sólo los agentes involucrados en la comunicación puedan descifrarlo y entenderlo.

En el caso de las comunicaciones informáticas, la criptografía ayuda a establecer comunicaciones seguras, tales comunicaciones deben garantizar las cuatro características siguientes:

- Confidencialidad. Los datos deben ser leídos sólo por los destinatarios.
- Autenticidad. Las entidades participantes deben ser perfectamente identificadas.
- Integridad. Asegurar que los datos transmitidos no han sido modificados.
- No repudio. Los componentes de la comunicación no pueden negar su participación.

Para conseguir las características anteriores se utiliza la criptografía en general y, las firmas digitales, funciones resumen o hash y certificados en particular (figura 7.1).

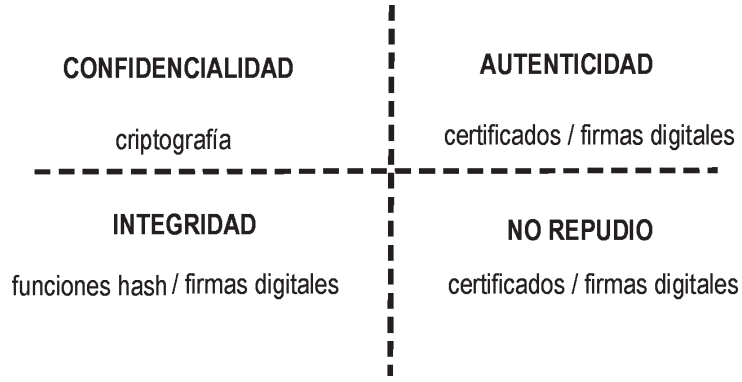


Figura 7.1. Características de la seguridad en las comunicaciones.

7.1.1. Cifrado simétrico

En este tipo de cifrado se utiliza una única clave tanto para cifrar como para descifrar los mensajes. La figura 7.2 muestra el proceso de cifrado y descifrado simétrico.

El algoritmo de cifrado tiene que ser robusto, es deseable de cara a garantizar la robustez del algoritmo que sea público. Si un algoritmo es público, todo el mundo lo puede analizar en busca de posibles debilidades, por tanto, si nadie encuentra debilidad alguna, el algoritmo se considera completamente robusto.

Tanto el emisor como el receptor deben conocer la clave de encriptación, tienen que obtenerla de forma segura y tienen que mantenerla en secreto, este es el principal problema de los algoritmos simétricos como se analizará posteriormente en este mismo capítulo.

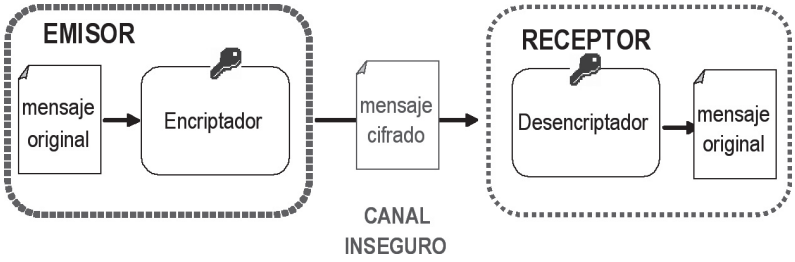


Figura 7.2. Esquema de cifrado simétrico

Los algoritmos simétricos sirven para la encriptación o cifrado en masa de datos o flujos de datos, tales datos pueden ser de tamaño constante o variable, depende de los algoritmos.

Independientemente de las bondades de cada algoritmo, características, número de etapas, etc. existen 2 tipos de algoritmos simétricos:

- Algoritmos por bloques. Son algoritmos de cifrado simétrico que trabajan dividiendo el mensaje a codificar en bloques de tamaño fijo, y aplican las distintas operaciones o etapas a cada uno de estos bloques.
- Algoritmos de flujo. Trabajan codificando el mensaje byte a byte utilizando generalmente operaciones xor.

Además, dentro de los algoritmos de cifrado por bloques podemos distinguir distintos modos de operación:

- Electronic CodeBook (ECB). Codifica por bloques independientes de 64 bits, todos con la misma clave.

- Cipher Block chaining (CBC). Los bloques del texto cifrado se relacionan mediante operaciones XOR.
- Output Feedback (OFB) Se realiza una OR-EXCLUSIVA entre caracteres o bits aislados del texto y las salidas del algoritmo. El algoritmo utiliza como entradas los textos cifrados.
- Cipher Feedback (CFB) Variante del anterior para mensajes muy largos.

Este tipo de algoritmos son más sencillos que el resto, por lo que, la principal ventaja que aportan es que son más rápidos, por esta razón son utilizados para codificar grandes cantidades de datos y prácticamente todas las comunicaciones.

El principal inconveniente de este tipo de algoritmos, tal y como se ha mencionado antes, es la distribución de las claves. Los agentes de la comunicación deben conocer la clave privada para cifrar, la clave se suele pasar por un canal inseguro, con el riesgo intrínseco de que alguien más (ajeno a la comunicación) pueda obtenerla. Los algoritmos simétricos se utilizan generalmente para dotar de confidencialidad a la comunicación.

Los algoritmos simétricos más utilizados son:

- DES
  - Algoritmo simétrico que codifica bloques de 64 bits con claves de 56 bits más 8 de paridad. Es de muy fácil implementación hardware y software.
  - Inconvenientes: dispone de una clave corta.
- TDES
  - Está basado en la aplicación de tres iteraciones del algoritmo DES, lo que equivale a cifrar con una clave más larga que en el algoritmo original.
  - Pueden realizarse con la misma o diferente clave.
  - Emplea claves de 128 bits de longitud (112 de clave y 16 de paridad).
- IDEA
  - Trabaja con bloques de texto de 64 bits y una clave de 128 bits. Puede trabajar con los 4 modos: ECB, CBC, CFB y OFB.
  - Opera con números de 16 bits utilizando operaciones como xor, suma de enteros o multiplicación de enteros.
- RC-5
  - Algoritmo de cifrado sucesor de los RC-2 y RC-4 de RSA Data Security, ampliamente configurable. Deja que sea el usuario

final el que decida la longitud de la clave, el tamaño del bloque de datos y el número de pasadas de cifrado.
- Es el algoritmo elegido por Netscape para la implementación del SSL.

Estos algoritmos se van a analizar con mayor nivel de detalle a continuación.

7.1.1.1. Algoritmo DES

El estándar de cifrado de datos o Data Encryption Standard (DES) es el algoritmo simétrico más utilizado en el mundo desde su aparición en 1977, gracias en gran medida a que fue adoptado como estándar de cifrado para comunicaciones no clasificadas por el gobierno de los EE.UU.

Aunque la validez teórica del algoritmo sigue vigente y no ha mostrado debilidad alguna, en el año 1998 un reto demostró que un ataque por fuerza bruta podía romper al cifrado mediante DES, debido principalmente a que utiliza una clave de 56 bits. Esta longitud de clave era buena en 1977, pero insuficiente para las capacidades de cómputo de 1998.

Como se ha mencionado antes, DES utiliza claves de 56 bits para codificar bloques con el tamaño de 64 bits. El algoritmo utiliza una red de Feistel (muy usada en algoritmos simétricos) de 16 rondas, calculando para cada una de las etapas una subclave distinta a partir de la clave de cifrado. Ejecuta también dos permutaciones, una al principio y otra al final del algoritmo.

Los principales problemas del algoritmo DES son:

- Algoritmo muy lento para tratarse de un algoritmo simétrico, debido principalmente al elevado número de rondas.
- Longitud de clave demasiado corta.
- Presenta el problema de las claves débiles. Claves que generan un conjunto de 16 subclaves iguales.

7.1.1.2. Algoritmo Triple-DES

Dado que la seguridad teórica del algoritmo DES sigue estando lo suficientemente probada y su problema real es la escasa longitud de clave, una alternativa perfectamente válida es utilizar el algoritmo pero con una longitud de clave mayor.

El algoritmo DES múltiple lo que hace es repetir el algoritmo DES varias veces utilizando en cada una de las pasadas la misma clave o claves distintas. En la práctica esta repetición del algoritmo es como usar una clave más larga, 112 bits para dos pasadas y 168 bits para tres pasadas. Tiene la ventaja adicional de seguir utilizando las implementaciones del algoritmo DES existentes, algunas de ellas en hardware.

De entre todos los posibles DES múltiples, Triple-DES es el más conocido y usado.

7.1.1.3. Algoritmo IDEA

El algoritmo de cifrado de datos internacional o International Data Encryption Algorithm (IDEA) es mucho más actual que DES, data de 1992. Al igual que DES, trabaja con bloques de 64 bits de longitud, pero emplea claves de 128 bits. También se utiliza el mismo algoritmo tanto para el cifrado como para el descifrado.

A diferencia del algoritmo anterior, IDEA no presenta el problema de las claves débiles, y su longitud de clave hace imposible un ataque por fuerza bruta con la capacidad de cómputo actual.

El funcionamiento del algoritmo IDEA consta de 8 etapas o rondas en vez de 16 que utiliza DES. Otra diferencia es que IDEA divide el bloque de datos en cuatro partes de 16 bits, mientras que en el caso anterior se divide en dos partes de 32 bits.

7.1.1.4. Algoritmo AES

En el año 1997 el NIST (Nacional Institute for Standards and Technology) convocó un concurso público para diseñar un algoritmo de cifrado simétrico que sustituyese al viejo DES. Después de un largo proceso y, previo paso por una final con 5 algoritmos, el NIST anunció oficialmente la adopción del algoritmo Rijndael como nuevo estándar de cifrado avanzado o Advanced Encryption Standard (AES).

AES es un sistema simétrico de cifrado por bloques diseñado para utilizar longitudes de clave y bloque variables, de entre 128 y 256 bits. Aunque el diseño del algoritmo permita longitudes variables, las implementaciones existentes usan una longitud de bloque de 128 bits, y una longitud de clave de 128, 192 o 256 bits.

AES no está formado por una red de Feistel, está formado por cuatro funciones invertibles diseñadas para resistencia frente a criptoanálisis

lineal y diferencial. Para descifrar aplicaremos en cada ronda las inversas de las funciones en orden contrario al usado en el cifrado.

Dado que tanto la longitud de la clave como de los bloques a cifrar son variables, tampoco existe un número de rondas fijo, ya que dicho número de rondas va a depender de las longitudes de clave y bloque seleccionadas inicialmente.

En cuanto a la seguridad del algoritmo, éste ha sido diseñado para resistir criptoanálisis lineal y diferencial. Además, y al contrario de su predecesor DES, no presenta problema de claves débiles ni semidébiles.

7.1.1.5. Algoritmo RC5

El cifrado de Rivest 5 o Rivest Cipher 5 (RC5) fue diseñado y desarrollado por Ron Rivest para la RSA Data Security. A diferencia de muchos esquemas, RC5 es configurable o variable en todos sus aspectos, puede utilizar distintos tamaños de bloque (32, 64 o 128 bits), permite tamaño variable de clave (entre 0 y 2040 bits) y número de vueltas entre 0 y 255. La combinación sugerida originalmente era bloques de 64 bits, claves de 128 bits y 12 vueltas. Es el algoritmo elegido por Netscape para su implementación de SSL.

7.1.1.6. Distribución de claves

La distribución de las claves es el principal problema de los cifrados simétricos, ya que, exceptuando el caso de que se pueda entregar la clave de cifrado físicamente, en general ésta se entregará pasando por algún medio o canal inseguro.

Para dos miembros A y B de una comunicación, existen varios tipos de distribución posible:

- A genera/selecciona una clave y se la entrega físicamente a B
- Un tercer miembro (C) genera/selecciona un clave y se la entrega físicamente a A y B
- Si A y B ya habían utilizado previamente una clave, uno de ellos le transmite al otro la nueva clave cifrándola con la antigua
- C transmite a A y a B la nueva clave mediante el uso de otra clave que los tres ya conocen (C es un centro de distribución de claves)

Posteriormente se estudiará un sistema de distribución de claves seguro basado en criptografía asimétrica e infraestructura de clave pública (PKI).

7.1.2. Cifrado asimétrico

Los algoritmos asimétricos o de clave pública fueron introducidos por Diffie y Hellman en 1976, se basan en disponer de distintas claves, concretamente pares de claves, como diferencia a los algoritmos anteriores donde sólo se dispone de una clave secreta. Una de las claves se denomina privada y la otra pública.

La clave privada sólo es conocida por el propietario de la clave, mientras que la clave pública es conocida por él y por todos los que quieran comunicarse con él de forma cifrada.

Además de la diferencia teórica del uso de dos claves en lugar de una, existen dos diferencias importantes entre los dos tipos de algoritmos. Una diferencia de los algoritmos asimétricos respecto a los simétricos es la longitud de clave, mientras que en un algoritmo simétrico, una clave de 128 bits se considera segura, en uno de clave pública se recomiendan claves de al menos 1024 bits. Otra diferencia es la rapidez de los algoritmos, los simétricos son considerablemente más rápidos que los asimétricos, debido principalmente a la complejidad de cálculo.

En el proceso de cifrado y descifrado también existe una diferencia fundamental, mientras que en la criptografía simétrica se hacen los dos procesos con la misma clave, en los algoritmos de clave pública se utiliza una clave para cifrar el mensaje y la otra para descifrarlo (figura 7.3).

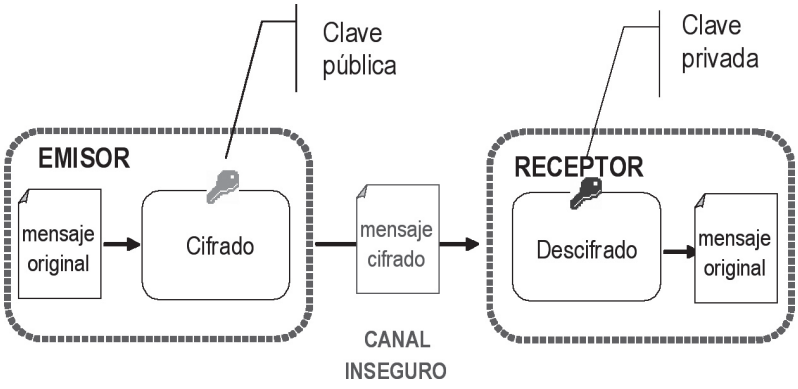


Figura 7.3. Esquema general del cifrado asimétrico.

Con este tipo de algoritmos se puede garantizar la integridad, confidencialidad, autenticación y no repudio en una comunicación.

Los algoritmos más comunes de este tipo son:

- DSS. Basado en el algoritmo DSA, creado para firmas digitales, permite claves sólo de 512 ó 1024 bits. Teniendo en cuenta lo indicado en párrafos anteriores, el tamaño de clave que utiliza se considera pequeño.
- RSA. Es el algoritmo asimétrico más utilizado (se usa en el protocolo SSH). Se basa en la dificultad para factorizar números grandes obteniendo las claves a partir de un número que se obtiene como producto de dos números primos grandes.

7.1.2.1. Algoritmo RSA

Uno de los primeros esquemas de clave pública, concretamente data del año 1977. Fue creado por Rivest, Shamir y Adleman, de ahí el nombre de RSA. Pese a ser de los primeros algoritmos asimétricos, se han desarrollado muchos, es uno de los más sencillos de implementar y está considerado como uno de los algoritmos asimétricos más seguros.

Su seguridad se basa en la dificultad computacional de factorizar números muy grandes. Las claves se obtienen a partir de un número que es el producto de dos números primos muy grandes. Concretamente, el proceso es el siguiente:

- Obtener aleatoriamente dos números primos muy grandes, p y q
- Calcular $n=p*q$
- Obtener e, primo relativo o coprimo con (p-1)(q-1), de forma que tenga inversa.
- Calcular d como la inversa de e
- (e,n) son la clave pública
- (d,n) son la clave privada
- Cifrado: $C = M^e \bmod n$, donde C es el mensaje cifrado y M es el mensaje original
- Descrifrado: $M = C^d \bmod n$

Para claves de 1024 bits es prácticamente inatacable, el atacante debe conocer p y q. El conocimiento de ambos es computacionalmente inabordable por tratarse de un problema exponencial, siempre que p y q sean números grandes.

7.1.3. Funciones HASH

Las funciones HASH son algoritmos que, dado un mensaje, lo transforman en un código de longitud fija. Se usa para proporcionar la llamada "huella digital" que identifica un mensaje. Las propiedades que debe presentar una función HASH para considerarla segura son:

- Dado un texto, siempre debe proporcionar la misma salida y debe ser única
- La función debe ser aleatoria y unidireccional
- La función tiene que poder ser aplicada a bloques de datos de cualquier tamaño
- La salida producida debe tener tamaño fijo
- El cálculo tiene que ser relativamente sencillo
- La función resumen tiene que ser una función que sea muy difícil o imposible encontrar inversa
- Función que sea difícil encontrar dos bloques de datos con el mismo resumen
- El principal uso que tienen es para proporcionar autenticidad e integridad
- Combinadas con técnicas de clave pública, se obtiene una forma eficiente de identificación (firmas digitales)

Los algoritmos HASH más usados son: MD5 y SHA

- **MD5**. Consiste en aplicar sucesivamente, a bloques de 512bits del mensaje, junto con 4 vectores de 32 bits cada uno, diversas operaciones lógicas. Tras cada iteración, se produce 128bits de salida (4 vectores de 32) que alimentan el proceso con el siguiente bloque del mensaje. La salida serán los 128bits resultado de la iteración con el último bloque. A cada bloque se le aplican 64 pasos.
- **SHA (SHA-1).** Algoritmo similar al MD5 con la diferencia de que se emplean vectores de 160bits por lo que la salida también tendrá 160bits. Con este algoritmo, cada bloque de 512bits del mensaje, sufre 80 pasos. Es más seguro que MD5, aunque también más lento.

7.1.3.1. Algoritmo SHA

Secure Hash Alrgorithm fue desarrollado por la NSA, publicado en 1993, y revisado en 1995 (SHA-1). Produce un resumen de 160 bits, y utiliza bloques del mensaje de 512 bits.

El algoritmo emplea cinco registros de 32 bits, en vez de cuatro como MD5, que deben ser inicializados antes de empezar con cinco valores hexadecimales fijos.

La parte principal del algoritmo está compuesta de cuatro iteraciones con 20 operaciones cada una. En estas cuatro iteraciones se emplean cuatro constantes, una por cada iteración, y tres funciones lógicas.

Al finalizar el último bloque, el valor de la función resumen será la concatenación de los cinco registros, proporcionando en total los 160 bits de salida del algoritmo SHA.

Como se ha mencionado anteriormente, el algoritmo SHA es muy seguro. La probabilidad de que dos mensajes tengan el mismo resumen es de $1/2^{20}$. La dificultar de encontrar un mensaje con un resumen determinado es de 2^{160} operaciones.

7.1.3.2. Firma digital

Las firmas digitales se componen de una función resumen y una clave asimétrica para garantizar la identidad de los miembros de una comunicación.

Los pasos que se deben de dar para crear una firma digital dependen de si se es emisor o receptor, en el lado del emisor se ejecutan las siguientes acciones:

- Creación del par de claves publica/privada para el emisor. Las firmas digitales se basan en las funciones resumen y el cifrado asimétrico. El cifrado de clave pública, como se ha comentado anteriormente en su apartado, se basa en pares de claves, donde la privada sólo la conoce el propietario, mientras que la pública la distribuye para que sea conocida por todos
- El emisor envía la clave pública al receptor. La clave pública permitirá al receptor descifrar los mensajes que le lleguen desde el emisor de la clave, pero también le permite enviar al emisor mensajes cifrados con la clave pública, que serán descifrados con la clave privada
- El emisor envía un mensaje al receptor, incluyendo la función resumen del mensaje, codificada con su clave privada, generando la firma digital. Es decir, la firma digital será la función resumen de un mensaje dado, cifrada con la clave privada

Por el contrario, en el lado del receptor, y asumiendo que ya conoce la clave pública del emisor, las acciones a llevar a cabo serán las siguientes:

- Receptor separa el mensaje en 2: documento original y firma
- Se descifra la firma digital mediante la clave pública del emisor, obteniendo la función resumen original del mensaje
- El receptor usa el mensaje original como entrada de la misma función resumen usada por el emisor, obteniendo la función resumen del mensaje en el lado receptor de la comunicación
- Se comprueba la validez: integridad y autenticación. Para comprobar la validez, el receptor compara las dos funciones resumen. En caso de coincidir, le está indicando que el mensaje es auténtico, y que no ha sufrido alteración durante la comunicación. Auténtico, ya que al descifrarlo con la clave pública del emisor, sólo ha podido ser cifrado con la clave privada, que sólo conoce el emisor y propietario de ambas claves. También se ha salvaguardado la integridad del mensaje, ya que de no ser así no habrían coincidido las dos funciones resumen.

El principal problema que presentan es el mismo que el de las claves asimétricas: la distribución de las claves y la "confianza" de las partes en que las claves públicas pertenecen a quien dicen ser. Para solucionar este problema se suelen utilizar terceras partes, e incluso las terceras partes pueden ser grupos, denominados anillos de confianza.

Cuando se utiliza la firma digital que permite autenticar al emisor y verificar la integridad del mensaje, el proceso se puede hacer de dos formas distintas:

- Hash cifrado
- Utilizando valor secreto

Cuando utilizamos hash cifrado, el emisor calcula la función resumen del mensaje a enviar, el resultado de la función resumen es entonces cifrado o codificado con la clave privada del emisor. La función resumen cifrada o firma digital se añade al mensaje y se transmiten al receptor.

El receptor divide el mensaje en dos partes: mensaje propiamente dicho y firma digital. Utilizará la clave pública del emisor para descifrar la firma digital y obtener la función resumen del mensaje original, calculará la función resumen del mensaje recibido, y comprobará que los dos resúmenes coinciden (figura 7.4).

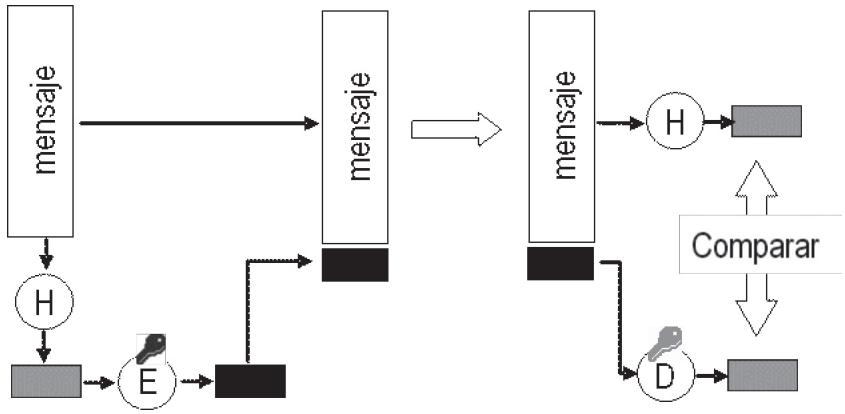


Figura 7.4. Cifrado mediante función hash.

El proceso es ligeramente distinto en el caso del hash mediante valor secreto. En este caso, la clave se añade al mensaje, y se obtiene la función resumen del conjunto, éste resumen se añade al mensaje original, y se transmite al receptor.

El receptor divide el mensaje en dos partes: mensaje y función resumen. Al mensaje le añade la clave y calcula la función resumen, comprobando que coincide con la recibida (figura 7.5).

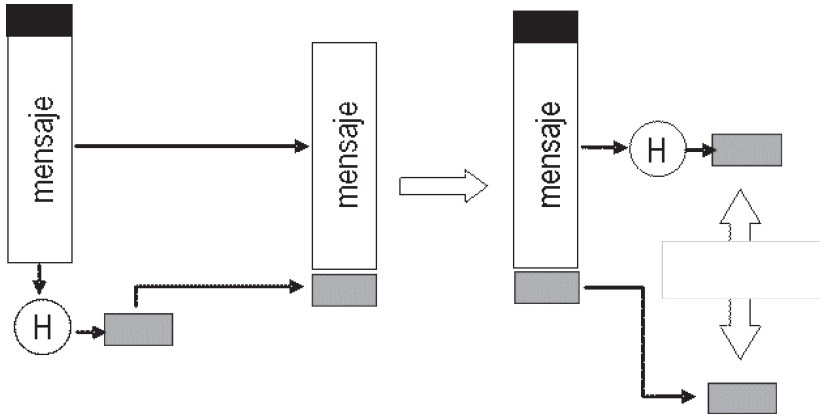


Figura 7.5. Función hash utilizando valor secreto.

Aunque no se ha comentado en ninguno de los dos casos, el mensaje debería transmitirse cifrado, y ser descifrado por el receptor. En este sentido, debemos obtener la firma del mensaje antes de codificarlo, ya que existen ataques que permiten manipular con éxito mensajes primero cifrados y luego firmados.

7.2. AUTENTICACIÓN

La autenticación es el proceso llevado a cabo para asegurarnos que un usuario o dispositivo es quien dice ser, y no está siendo suplantado por otro. El objetivo es verificar la autenticidad de los mensajes en una transmisión, comprobar la identidad de un usuario en una conexión, e incluso, la autenticidad de un dispositivo.

Aunque vamos a abordar en más profundidad la autenticación en las comunicaciones basada en herramientas analizadas anteriormente, también mostraremos la autenticación de usuarios mediante contraseñas y la identificación de dispositivos mediante protocolos de desafío.

En general, y aunque no vamos a abordar todos los posibles mecanismos de autenticación, se pueden utilizar las siguientes herramientas para proporcionar autenticación:

- Firmas digitales
- Certificados digitales
- Contraseñas
- Características biométricas

Las firmas digitales las hemos abordado en el apartado anterior, consisten en una secuencia de información que se añade al mensaje para demostrar su autenticidad. Tal como hemos visto, el texto añadido es el resultado de aplicar al mensaje una función resumen y cifrar el resumen con la clave.

El receptor descifra la firma digital y comprueba las dos funciones hash, si son iguales se garantiza la autenticidad y la integridad. El hecho de que el receptor haya descifrado la firma digital con la clave pública del emisor, demuestra que sólo ha podido ser firmado con la clave privada correspondiente, autenticando de esta forma el mensaje como perteneciente realmente al emisor.

Un certificado digital es, fundamentalmente, una clave pública cifrada digitalmente por una *autoridad certificadora*. De esta manera, la autoridad

certifica que la clave pública pertenece al usuario. Previamente, la entidad de certificación tendrá que comprobar que la clave pública es auténtica.

Con la utilización de certificados digitales, los miembros de una comunicación pueden estar seguros de la autenticidad del resto, ya que ha sido comprobada por una autoridad competente. Se pueden generar certificados sin ser cifrados por una autoridad de certificación. En este caso, el destinatario es el responsable de aceptar o rechazar el certificado. En España, la autoridad certificadora es la Fábrica Nacional de Moneda y Timbre.

En la autenticación de las comunicaciones se pueden introducir ayudas adicionales, por ejemplo, incluir en el mensaje su número de secuencia y marcas de tiempo, y comprobar la secuencia relativa a otros mensajes.

El mecanismo más utilizado para verificar la identidad de un usuario cuando se conecta a un sistema es el uso de contraseñas o passwords. Con este método, el sistema informático almacena combinaciones de nombres de usuario y contraseñas asociadas. Cuando un usuario quiere acceder al sistema, introduce su nombre de usuario, el sistema le pedirá la contraseña, y finalmente comprobará que la contraseña introducida corresponde con el usuario concreto.

Desde el punto de vista de la seguridad, debemos tener en cuenta que, si el sistema informático guarda las contraseñas de los usuarios en texto plano, cualquiera que acceda al archivo conocerá las contraseñas de todos los usuarios y podrá suplantar a cada uno de ellos. Debido al aspecto anterior, los sistemas no guardan las contraseñas directamente, sino que almacenan el resultado de aplicar a la contraseña una función resumen. De esta forma, si alguien accede al archivo de contraseñas, le resultará muy difícil obtenerlas debido a las propiedades de las funciones resumen. Este es el esquema utilizado por el sistema operativo UNIX.

Para la autenticación de usuarios se pueden utilizar también sus propias características biométricas. Las huellas dactilares son un mecanismo utilizado para la autenticación en muchos sistemas, no es de extrañar que muchos portátiles lleven lectores de huellas dactilares.

Finalmente, existen situaciones en las que el objeto a ser autenticado no es un usuario sino un dispositivo, este tipo de autenticación se presenta sobre todo en la computación móvil: teléfonos móviles, pdas, etc. Los protocolos que permiten verificar la identidad de un dispositivo se basan en el establecimiento de un desafío.

Cuando un dispositivo quiere autenticarse frente a un sistema informático, el sistema genera un desafío, típicamente un valor o texto aleatorio. El sistema envía el desafío al dispositivo, el dispositivo calculará el código de

autenticación del mensaje o MAC (Message Autentication Code), según el algoritmo que tenga implementado y una clave secreta K, y devolverá dicho código al sistema. El sistema calculará a su vez el MAC empleando la misma clave, y comprobará si coincide con el valor recibido desde el dispositivo.

Obviamente, tanto el dispositivo como el sistema deben conocer la clave secreta, esta clave la establecerán ambos cuando el dispositivo se da de alta en el sistema. Observemos que el proceso de autenticación se produce cada vez que el dispositivo desea conectarse con el sistema.

Este tipo de autenticación lo emplean, por ejemplo, las tarjetas SIM de los teléfonos móviles GSM y de tercera generación 3G. La clave secreta se calcula cuando se activa la tarjeta SIM, en este caso tiene una longitud de 128 bits. La mayoría de implementaciones GSM usan como algoritmo MAC un algoritmo denominado COMP-128, que presenta una seguridad relativamente débil. Como diferencia fundamental entre la autenticación GSM y 3G podemos destacar que, la autenticación GSM sólo se produce en un sentido, mientras que en la telefonía móvil de tercera generación, la red del proveedor de servicios también se autentica respecto al dispositivo.

7.3. INTEGRIDAD

La característica de integridad persigue garantizar que el contenido del mensaje no ha sido alterado durante la comunicación, asegurando que el mensaje enviado por el emisor es exactamente el mismo que ha recibido el receptor.

Los principales mecanismos disponibles para garantizar la integridad son:

- Uso de checksums
- Firmas digitales
- IPSec

La forma más sencilla de comprobar la integridad es mediante el uso de checksums. Aplicar al mensaje una pequeña fórmula que devuelve un código, éste código se transmite con el mensaje. El receptor aplicará al mensaje recibido la misma fórmula, esperando obtener el mismo código. Si el código del mensaje coincide con el calculado por el receptor, tenemos una alta probabilidad de que el mensaje no ha sufrido alteraciones en la comunicación, y por tanto se haya mantenido su integridad. Los dos checksums más usados son los *bits de paridad* y el *chequeo de redundancia cíclica* o CRC.

CRC utiliza polinomios matemáticos para dividir la entrada, y el resto representa el código CRC. CRC-32 es un código de redundancia cíclica donde se ha utilizado un polinomio de 32 términos. CRC-32 se usa, por ejemplo, en algoritmos de compresión de archivos.

El problema de los códigos CRC es que se puede conocer el cambio que se producirá en el código, ante cambios producidos en el mensaje. Por lo tanto, constituye un mecanismo válido para errores reales o no intencionados en la transmisión, pero no para cambios totalmente intencionados producidos de forma maliciosa.

En el apartado anterior, hemos visto como utilizar las firmas digitales para aportar autenticación en la comunicación. El mismo mecanismo, y sin modificar para nada el proceso, nos sirve para garantizar la integridad del mensaje.

El emisor añade al mensaje la firma digital, el receptor descifra la firma digital y calcula la función resumen del mensaje recibido, comprueba las dos funciones hash, si son iguales se garantiza la autenticidad y la integridad. El hecho de que el resultado de la función hash calculada y enviada por el emisor, y la calculada por el receptor coincidan, garantiza que el mensaje no ha sido alterado durante la transmisión. Por supuesto, esta garantía es probabilística, depende de la probabilidad que tenga la función resumen de generar el mismo resumen para dos mensajes distintos. En el algoritmo SHA-1 sabemos que la probabilidad de que dos mensajes tengan el mismo resumen es de $1/2^{20}$.

La última herramienta que vamos a ver para aportar integridad en las comunicaciones es el uso del conjunto de protocolos IPSec. IPSec está formado principalmente por tres protocolos:

- Intercambio de Claves de Internet o IKE (Internet Key Exchange). Protocolo para la gestión de claves.
- Cabecera de Autenticación o AH (Authentication Header). Componente para proporcionar integridad al mensaje.
- Carga con Encapsulado de Seguridad o ESP (Encapsulating Security Payload). Diseñado para aportar confidencialidad.

Como hemos mencionado, el protocolo AH proporciona integridad al mensaje calculando un valor de chequeo de integridad ICV (Integrity Check Value) asociado a cada paquete, para el cálculo puede utilizar tanto MD5 como SHA-1. El receptor del mensaje calcula el ICV del paquete, si coinciden los dos valores, aceptará el paquete.

7.4. CONFIDENCIALIDAD

Los datos del mensaje deben ser leídos sólo por los destinatarios, o lo que es lo mismo, no deben poder ser leídos por cualquiera que intercepte el mensaje. Obviamente, no podemos garantizar que el mensaje no será interceptado, pero sí podemos asegurar que, si el mensaje es interceptado no será leído y entendido.

La criptografía es el mecanismo utilizado para garantizar la confidencialidad del mensaje, y asegurar que el mensaje no sea descifrado por terceras partes. El emisor antes de enviar el mensaje lo cifrará utilizando una clave, el receptor lo descifrará usando la clave. La clave será la misma si usamos criptografía simétrica, o distinta si utilizamos algoritmos asimétricos o de clave pública.

Los algoritmos simétricos son los más utilizados para el cifrado de mensajes, debido a su probada seguridad y a su rapidez para cifrar. Como hemos comentado al abordar el tema de los algoritmos simétricos, su principal problema es el intercambio de claves. El emisor y el receptor, pero sólo ellos, deben conocer la clave de cifrado, para ello la tienen que intercambiar previamente por un canal inseguro.

Los algoritmos asimétricos o de clave pública se pueden usar perfectamente para cifrar y descifrar mensajes, pero presentan el problema de ser más lentos que los simétricos y necesitar más carga computacional. En este tipo de algoritmos, los mensajes se cifran con la clave privada y se descifran con la pública, y viceversa.

En la realidad, se suelen utilizar los algoritmos de clave pública para establecer una clave de sesión, así podremos acordar una clave mediante un canal seguro. Finalmente, se usa la clave de sesión establecida, para cifrar el mensaje utilizando un algoritmo simétrico. De esta manera, se consigue el beneficio de intercambiar la clave por un canal seguro y la rapidez a la hora de cifrar el mensaje.

7.5. CAPA DE SOCKETS SEGURA

SSL son las iniciales de capa de sockets segura (Secure Sockets Layer), proporciona una capa de transporte segura con TCP entre un cliente y un servidor. Otras tecnologías de este nivel son TLS y SSH.

SSL se introduce en la pila de protocolos de TCP/IP como un protocolo de propósito general que permite un flujo de datos confiable entre una aplicación y el nivel de transporte, permitiendo autenticación y no repudio en el lado del servidor mediante certificados digitales (X.509).

Además de autenticación, proporciona integridad y confidencialidad de los datos, de tal forma que con el uso de SSL podemos cumplir o garantizar las tres características de seguridad básicas en las comunicaciones.

Mediante el uso de SSL se pueden establecer comunicaciones seguras, para ello se deben seguir una serie de pasos:

- Se debe hacer una solicitud de seguridad.
- Después de realizar la solicitud, se deben establecer los parámetros de funcionamiento que se van a utilizar con SSL. Esta fase se conoce como SSL Handshake o saludo SSL.
- Una vez se haya establecido una comunicación segura, se deben hacer verificaciones periódicas para garantizar que la comunicación sigue siendo segura a medida que se transmiten datos
- Finalmente, tras completar la transacción, se tiene que terminar correctamente la conexión segura.

Los pasos del proceso analizado en el párrafo anterior no se llevan a cabo por el mismo módulo, sino que SSL está formado por distintos subsistemas o protocolos, donde cada uno se encarga de aspectos específicos del proceso de comunicación descrito.

7.5.1. Arquitectura SSL

La arquitectura interna de SSL está formada por cuatro subsistemas o módulos, como se puede apreciar en la figura 7.6. Desde el punto de vista

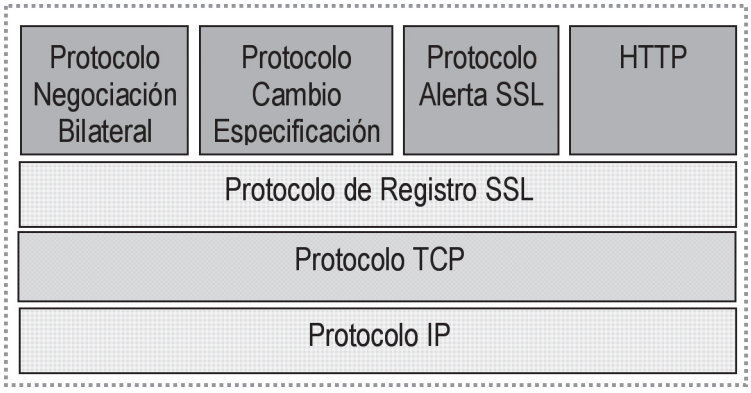


Figura 7.6. Subsistemas del protocolo SSL.

de la arquitectura, debemos observar el nivel o la capa donde se coloca el módulo principal de SSL. El protocolo de registro SSL, encargado de la integridad y confidencialidad de la conexión, está en una capa intermedia entre la capa de transporte y la capa de aplicación de la pila de protocolos TCP/IP.

Cada uno de los subsistemas o módulos anteriores tiene una función distinta que se aborda en los apartados siguientes.

7.5.1.1. Protocolo de negociación bilateral

Complejo protocolo que permite a los componentes de la conexión autenticarse mutuamente y negociar los parámetros de la sesión. Se encarga de iniciar, mantener y terminar las conexiones SSL.

Con el fin de establecer los parámetros y configurar la sesión, el cliente y el servidor intercambian información de la versión de SSL que tienen implementada, los algoritmos de cifrado, las funciones resumen y las longitudes de clave máximas que admiten para cada uno de ellos.

Los distintos tipos de cifrado que se pueden utilizar en SSL son:

- **RSA + Triple DES de 168 bits + SHA-1**. Soportado por las versiones 2.0 y 3.0 de SSL, es uno de los conjuntos más fuertes en cuanto a seguridad, ya que son posibles $3.7 * 10^{50}$ claves simétricas diferentes, por lo que es muy difícil de romper. Por ahora sólo está permitido su uso en Estados Unidos, aplicándose sobre todo en transacciones bancarias.
- **RSA + RC4 de 128 bits + MD5**. Soportado por las versiones 2.0 y 3.0 de SSL, permite $3.4 * 10^{38}$ claves simétricas diferentes que, aunque es un número inferior que el del caso anterior, da la misma fortaleza al sistema. Análogamente, en teoría sólo se permite su uso comercial en Estados Unidos, aunque actualmente ya es posible su implementación en los navegadores más comunes, siendo usado por organismos gubernamentales, grandes empresas y entidades bancarias.
- **RSA + RC2 de 128 bits + MD5.** Soportado sólo por SSL 2.0, permite $3.4 * 10^{38}$ claves simétricas diferentes, y es de fortaleza similar a los anteriores, aunque es más lento a la hora de operar. Sólo se permite su uso comercial en Estados Unidos, aunque actualmente ya es posible su implementación en los navegadores más comunes.

- **RSA + DES de 56 bits + SHA-1**. Soportado por las versiones 2.0 y 3.0 de SSL, aunque es el caso de la versión 2.0 se suele usar MD5 en vez de SHA-1. Es un sistema menos seguro que los anteriores, permitiendo $7.2 * 10^{16}$ claves simétricas diferentes, y es el que suelen traer por defecto los navegadores web en la actualidad (en realidad son 48 bits para clave y 8 para comprobación de errores).
- **RSA + RC4 de 40 bits + MD5**. Soportado por las versiones 2.0 y 3.0 de SSL, ha sido el sistema más común permitido para exportaciones fuera de Estados Unidos. Permite aproximadamente $1.1 * 10^{12}$ claves simétricas diferentes, y una velocidad de proceso muy elevada, aunque su seguridad es ya cuestionable con las técnicas de Criptoanálisis actuales.
- **RSA + RC2 de 40 bits + MD5**. Análogo al sistema anterior, aunque de velocidad de proceso bastante inferior.
- **Sólo MD5**. Usado únicamente para autentificar mensajes y descubrir ataques a la integridad de los mismos. Se usa cuando el navegador cliente y el servidor no tienen ningún sistema SSL común, lo que hace imposible el establecimiento de una comunicación cifrada. No es soportado por SSL 2.0, pero si por la versión 3.0.

Además de negociar los algoritmos de cifrado y funciones hash, el cliente solicita al servidor el envío de su Certificado Digital X.509 v3, con objeto de verificar la identidad del mismo y recoger su clave pública. Opcionalmente, el servidor puede solicitar certificado digital al cliente.

Finalmente, después de establecer los algoritmos de cifrado, el cliente genera la clave simétrica a utilizar en la comunicación y se la envía al servidor.

7.5.1.2. Protocolo de cambio de especificación de cifrado

Protocolo simple que actualiza el repertorio de cifrado, estableciendo las transiciones entre distintas políticas de cifrado a utilizar en cada conexión, e incluso cambios que se puedan producir en el transcurso de la sesión.

7.5.1.3. Protocolo de alerta

Protocolo usado para el transporte de alertas SSL. Se encarga de enviar mensajes de aviso de los problemas que ocurren durante la conexión, si el problema es leve se puede solucionar cambiando los parámetros, si es grave puede incluso llegar a terminar la sesión.

7.5.1.4. Protocolo de registro

Asegura la autenticación, integridad y confidencialidad de cada uno de los bloques que forman el mensaje, teniendo en cuenta que no se permiten bloques mayores de 16K bytes.

La autenticación y la integridad quedan garantizadas mediante el uso de firmas digitales, utilizando la función resumen y la clave establecidas en el protocolo de negociación bilateral.

La confidencialidad se aporta cifrando toda la información intercambiada en la comunicación, para cifrar se utilizan el algoritmo simétrico y la clave acordadas en el protocolo de negociación bilateral.

7.5.2. Seguridad en la capa de transporte

El protocolo de seguridad en la capa de transporte o Transport Layer Security (TLS) es una evolución de SSL versión 3. El objetivo de su desarrollo era disponer de un mecanismo para establecer comunicaciones seguras a través de Internet, pero que fuera completamente público, SSL es propiedad de Netscape. Asimismo, perseguía solucionar algunas de las deficiencias que presentaba SSL. TLS es un protocolo público desarrollado por la IETF totalmente compatible con SSL.

Desde el punto de vista de la arquitectura, TLS es igual que SSL. Existen los mismos módulos o protocolos que existían en SSL donde, igual que en el protocolo anterior, el Protocolo de Mutuo Acuerdo TLS y Protocolo de Registro TLS constituyen la base de funcionamiento del conjunto.

Aunque fundamentalmente es igual que SSL, existen algunas diferencias:

- TLS no soporta el algoritmo de cifrado simétrico Fortezza, por tratarse de un algoritmo de propiedad privada.
- Calcula las claves de sesión de forma distinta.
- TLS utiliza cinco campos para el cálculo del MAC, mientras que SSL usa tres. Esta diferencia permite a TLS aumentar la seguridad en el tratamiento de la autenticación y la integridad.

7.5.3. Certificados X.509

Como hemos mencionado con anterioridad, un certificado digital permite garantizar con confianza la identidad de una persona u organización que interviene en una comunicación. Actualmente se utiliza el estándar X.509 v3.

Una empresa de confianza (Autoridad Certificadora) es la encargada de realizar el certificado. Dos empresas certificadoras de reconocido prestigio internacional son Verisign y Thawte, a nivel nacional la entidad certificadora es la Fábrica Española de Moneda y Timbre.

El certificado incorpora la llave pública firmada digitalmente, que se suele usar para atestiguar la validez de la clave de un elemento de una comunicación.

Aunque los campos más importantes de un certificado digital son el identificador o número de serie y la clave pública firmada, un certificado X.509 versión 3 tiene otros campos que abordamos a continuación:

- Número de versión. Número de versión del certificado codificado, puede tener los valores 1, 2 y 3, generalmente tendrá el valor 3.
- Número de serie del certificado o Identificador. Es un valor único asignado por la autoridad de certificación.
- Información del algoritmo del emisor. Informa sobre el algoritmo usado para firmar el certificado, por ejemplo, RSA.
- Emisor del certificado o nombre de la CA. Campo donde se identifica la autoridad certificadora CA que ha firmado y emitido el certificado.
- Validez. Plazo de tiempo durante el cual el certificado es válido, fecha inicial y final.
- Nombre del sujeto. Identidad del propietario del certificado, el nombre debe ser único.
- Información clave pública. Campo que contiene la clave pública del sujeto firmada digitalmente.
- Firma digital de la CA. Firma digital de la autoridad certificadora con el fin de que cualquiera pueda comprobar la autenticidad del firmante del certificado.

Como hemos visto, con el sistema de certificados, se introduce también el concepto de autoridad certificadora, que será el tercero de confianza en la comunicación, responsabilizándose de la validez de los certificados.

Pero, ¿cuál es el ámbito de aplicación o alcance del certificado digital?, es decir, si emite el certificado una entidad local, dicho certificado puede ser utilizado fuera del ámbito de la entidad local. El certificado emitido por una autoridad podrá ser utilizado fuera de su ámbito, siempre y cuando la entidad certificadora correspondiente esté a su vez certificada por otra autoridad de nivel superior. Se establece de esta manera una jerarquía de confianza entre autoridades de certificación.

Mediante las jerarquías de confianza se puede verificar el certificado digital de cualquier autoridad certificadora ajena, a través de otra entidad certificadora de nivel superior. De esta manera, podemos aceptar los certificados emitidos por cualquier entidad. Debemos tener en cuenta que una autoridad de certificación suele tener un ámbito muy local, por ejemplo, un campus universitario como el de la Universidad de Alicante (figura 7.7).

Desde el punto de vista de la seguridad, una estructura jerárquica presenta el problema de poseer un "único punto de fallo", es decir, si se compromete una CA de un nivel superior, se deben revocar todos los certificados emitidos por ésta autoridad y sus subordinadas.

Además de la estructura jerárquica de organización de las CA, se pueden utilizar enfoques distribuidos, denominados *anillos de confianza,* donde cada sujeto es una CA. Mediante anillos de confianza se obtiene mayor seguridad, pero es mucho más complejo de gestionar. Suelen ser esquemas utilizados en redes sin infraestructuras o Ah Hoc.

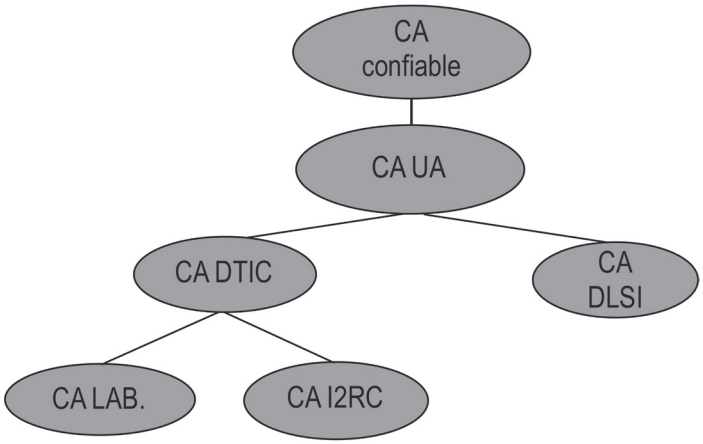


Figura 7.7. Jerarquías de confianza.

A todo el conjunto de protocolos y servicios que soportan aplicaciones basadas en criptografía de clave pública se denomina infraestructura de clave pública o PKI (Public Key Infraestructura). La PKI está basada en la confianza en terceras partes, es decir, en las autoridades de certificación. Más concretamente, la PKI se materializa en los certificados digitales y los protocolos que hacen uso de ellos, como SSL o TLS.

La PKI permite la creación y distribución de claves, establece mecanismos de revisión y revocación, así como, la gestión de los niveles de confianza entre autoridades certificadoras.

7.6. CASO DE USO: OPENSSL

Librería de código abierto que implementa todo el protocolo SSL y la infraestructura de clave pública. Existen varias empresas que tienen implementado software PKI, Computer Asociates ha desarrollado el paquete eTrust PKI, Red Hat tiene el software Netscape Certificate Management System, Microsoft, RSA Security, etc. Estudiaremos OpenSSL por tratarse de un paquete libre y de código abierto.

El paquete OpenSSL contiene herramientas de administración y librerías relacionadas con la criptografía. Son útiles para suministrar funciones criptográficas a otros paquetes de la capa de aplicación, cualquier aplicación en general y, OpenSSH y navegadores web (para acceso seguro a sitios https) en particular.

El contenido del paquete OpenSSL está formado por tres componentes principales:

- openssl. La herramienta *openssl* se usa desde la línea de comandos para utilizar las funciones criptográficas de la librería.
- crypto. En esta librería están implementados todos los algoritmos criptográficos que soporta el paquete, funciones de cifrado y resumen, generadores de claves y generadores pseudoaleatorios. La librería es utilizada por el propio OpenSSL en la implementación de SSL, TLS y S/MIME.
- ssl. Esta librería implementa los protocolos SSL y TLS. Y contiene una completa API que puede ser usada por otras aplicaciones de nivel superior.

En las siguientes secciones vamos a abordar los distintos componentes del paquete OpenSSL con mayor detenimiento.

7.6.1. Instalación de OpenSSL

El paquete OpenSSL está disponible tanto para Windows como para Linux, en nuestro caso abordaremos la instalación del paquete en Linux. No obstante, la instalación en Windows es tan sencilla como descargarse un autoejecutable, ejecutarlo y pulsar el botón *siguiente* en seis ocasiones.

Los pasos a seguir para instalar el paquete en Linux se muestran en el listado 7.1.

```
# tar -xvzf openssl-0.9.8e.tar.gz

# cd openssl-0.9.8e

# ./config

# make

# make test

# make install
```

Listado 7.1. Instalación de OpenSSL

Como podemos observar, la instalación es muy sencilla y parecida a la mayoría de instalaciones en Linux. Lo primero que tenemos que hacer es descargar la última versión del paquete, en este momento —mayo 2007— la última versión es la 0.9.8e. Descomprimimos y desempaquetemos la distribución, y nos situamos en el directorio que habrá creado.

Después ejecutaremos el script *config*, que obtendrá información relativa al sistema y configurará otros scripts de instalación del paquete. Mediante la orden *make* compilaremos el código fuente de OpenSSL. El penúltimo paso es testear el resultado de la compilación, para asegurarnos que ésta ha sido válida y no contiene errores. Finalmente, si el test se ha resuelto con éxito, pasamos a instalar los binarios del paquete OpenSSL generados en la compilación.

7.6.2. Comandos

En este apartado vamos a analizar la herramienta de línea de comandos *openssl*. Como hemos comentado anteriormente, este programa permite utilizar las funciones criptográficas del paquete desde la línea de comandos. Concretamente, la herramienta ofrece los siguientes servicios:

- Creación de claves RSA, DH y DSA
- Creación de certificados X.509

- Cálculo de resúmenes de mensajes
- Cifrar y descifrar con los distintos algoritmos
- Realizar test del cliente y servidor SSL/TLS
- Manejo de correos electrónicos firmados o cifrados S/MIME

Aunque, como es natural, no vamos a abordar todos los comandos que permite ejecutar la herramienta, si analizaremos los que nos permiten llevar a cabo las principales funciones. Además, estos comandos reciben un amplio abanico de opciones y argumentos que tampoco analizaremos, para ello se puede consultar la página del manual correspondiente. El objetivo es describir los comandos fundamentales.

- openssl list-standard-commands. Muestra una lista de todos los comandos estándar que podemos ejecutar.
- openssl ca. Este comando es una autoridad de certificación mínima, permite atender peticiones y generar certificados firmados. Mantiene una base de datos con los certificados vigentes y su estado.
- openssl crl. Parecido al comando anterior, en este caso permite la gestión de la lista de revocación de certificados.
- openssl dgst. Comando utilizado para calcular el mensaje resumen de un archivo o archivos, recibe como parámetro la función hash a utilizar. También se puede usar para generar la firma digital del archivo.
- openssl enc. Permite el cifrado y descifrado de archivos utilizando los distintos algoritmos simétricos soportados por la herramienta.
- openssl s_client. Implementa un cliente SSL/TLS genérico que puede establecer una conexión segura con un servidor.
- openssl s_server. Servidor SSL/TLS genérico que puede aceptar conexiones seguras desde clientes remotos.
- openssl smime. Comando para el procesamiento de correo electrónico seguro S/MIME.
- openssl x509. Con este comando podremos gestionar los datos de los certificados, mostrar los datos, convertirlos a otros formatos, etc.

Finalmente, existen muchos comandos para la generación de claves y secuencias pseudoaleatorias, y la gestión de los parámetros de los distintos algoritmos.

7.6.3. Librería

Hemos abordado en el apartado anterior el paquete OpenSSL desde el punto de vista de su utilización en la línea de comandos. Sin embargo, la herramienta ofrece, además de los comandos, una extensa librería con la que poder implementar cualquier programa que haga uso tanto de algoritmos criptográficos como de funciones para establecer conexiones seguras vía SSL o TLS.

Desde el punto de vista de su utilización como API de programación, el paquete OpenSSL ofrece dos librerías distintas: *crypto* y *ssl*. C*rypto* es una librería que implementa todos los algoritmos criptográficos; cifrados simétricos, asimétricos o de clave pública, funciones resumen, generación de secuencias pseudoaleatorias y funciones de nivel superior. La librería s*sl* permite ejecutar funciones que implementan los protocolos de establecimiento de conexiones seguras SSL y TLS.

Crypto se puede considerar de más bajo nivel, ya que es utilizada tanto por las funciones de la librería ssl como por cualquier aplicación que queramos implementar. Implementa un amplio rango de algoritmos criptográficos usados en estándares de Internet como los propios SSL, TLS y S/MIME, y también se utiliza para implementar SSH y OpenPGP.

Crypto esta formada por distintas sublibrerías que implementan los algoritmos individuales pertenecientes a las categorías siguientes:

- Cifrados simétricos. Dentro de este tipo de algoritmos tenemos las librerías *blowfish*, *cast*, *des*, *idea*, *rc2*, *rc4* y *rc5*. Cada una de estas sublibrerías está constituida por una API que contiene las funciones necesarias para la implementación del algoritmo criptográfico concreto.
- Criptografía de clave pública. Implementa las funciones correspondientes a los algoritmos *rsa*, *dsa* y *dh*.
- Certificados. Sublibrería *x509* con las funciones que nos permiten hacer uso de los certificados digitales en nuestras aplicaciones.
- Funciones resumen. Principalmente, incluye las librerías *sha* y *md5*, con las operaciones necesarias para implementar resúmenes de archivos.
- Otras funciones. Dentro de esta categoría son destacables la librería *rand* que permite generar números pseudoaleatorios, y *evp* que se utiliza como interfaz de alto nivel para acceder a librerías de algoritmos que no pueden ser invocadas directamente.

La librería ssl implementa los protocolos SSL y TLS estudiados en apartados anteriores. En la versión actual, el API está constituida por 214 funciones, de ellas, las que implementan el saludo, la transferencia y el cierre son las siguientes:

- SSL_accept. Función utilizada por un servidor para esperar que un cliente inicie un saludo SSL/TLS.
- SSL_connect. Usada por un cliente para iniciar un protocolo de saludo SSL/TLS con un servidor.
- SSL_read. Permite leer bytes desde la conexión segura.
- SSL_write. Función que se usa para escribir bytes en la conexión segura.
- SSL_shutdown. Esta rutina permitirá a las aplicaciones cerrar conexiones seguras.

Finalmente indicar que, en la librería ssl el archivo de cabecera que se debe incluir es siempre ssl.h. Por el contrario, en la librería crypto existen muchos archivos de cabecera, que coinciden con el nombre del algoritmo, por ejemplo des.h.

8. SISTEMA DE ARCHIVOS DISTRIBUIDO

Con el desarrollo de la informática y las redes, los nodos han dejado de ser entidades aisladas sin conexión con el resto, para convertirse en parte de una entidad mayor que es la red. De igual manera, los sistemas de archivos han pasado de ser exclusivamente locales, a sistemas remotos o distribuidos compartidos por los computadores de la red.

Los sistemas de archivos remotos permiten gestionar desde un nodo cliente un sistema de archivos externo, o dicho de otra manera, permiten utilizar un sistema de archivos remoto como si de un sistema local se tratarse. Estos sistemas suelen estar basados en una arquitectura cliente-servidor, donde el servidor reside en el computador que exporta o comparte su sistema de archivos. La aplicación cliente solicitará el acceso a la información a través del servicio, que accederá al sistema de archivos y devolverá al cliente la información solicitada (figura 8.1).

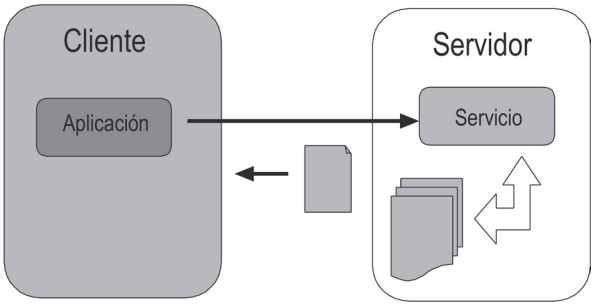


Figura 8.1. Arquitectura cliente-servidor en los sistemas de archivos remotos.

La utilización de sistemas de archivos distribuidos persigue una serie de objetivos que se indican a continuación:

- Acceso a archivos comunes desde múltiples clientes
- Compartir información
- Almacenar información masiva, optimizando del espacio de almacenamiento
- Acceso a archivos en equipos sin unidades de almacenamiento
- Cuentas de usuario centralizadas (independencia del equipo de trabajo)

8.1. TRANSFERENCIA DE ARCHIVOS VS. ARCHIVOS DISTRIBUIDOS

A la hora de compartir archivos se distingue entre sistemas de transferencia de archivos y sistemas de archivos distribuidos. En los sistemas de transferencia, el cliente accede al servidor para la transferencia del archivo a la máquina local del cliente, a partir de ese instante el cliente puede trabajar con la copia del archivo que tiene en su máquina local. Desde el punto de vista de la arquitectura, tanto el cliente como el servidor se ejecutan en el nivel de aplicación. Además, cada aplicación cliente que necesite acceso a los archivos remotos tiene que implementar el protocolo de transferencia, siendo responsabilidad de la propia aplicación y no del sistema operativo la transferencia de la información (figura 8.2).

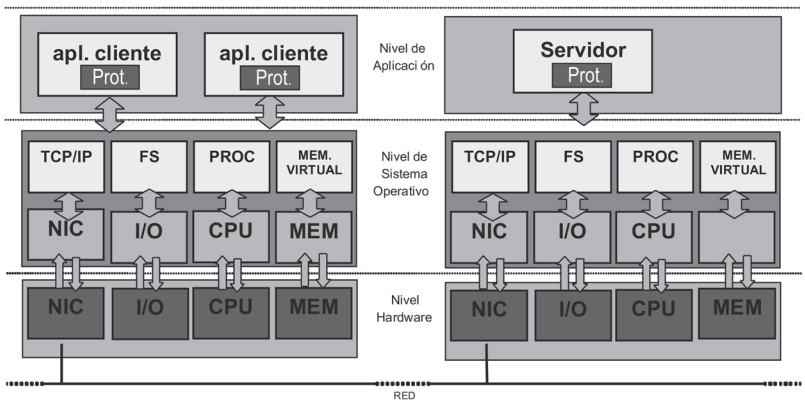


Figura 8.2. Arquitectura de los sistemas de transferencia de archivos

La ventaja de los sistemas de archivos distribuidos es que el protocolo de transferencia de la información está integrado en el sistema operativo, funciona por tanto de forma transparente a las aplicaciones de usuario, siendo la transferencia de la información responsabilidad del sistema, no de la aplicación (figura 8.3).

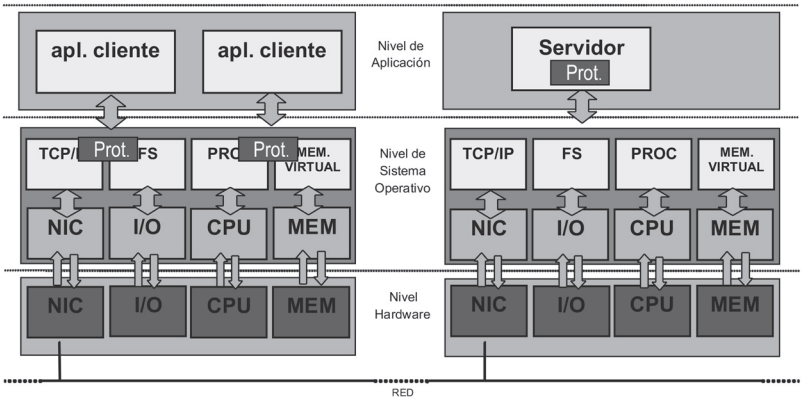


Figura 8.3. Arquitectura de los sistemas de archivos distribuidos

8.2. EL SISTEMA DE ARCHIVOS VIRTUAL

Varios de los sistemas de archivos distribuidos que veremos a continuación están integrados en el Virtual File System (VFS) de Linux, lo cual proporciona una interfaz homogénea de acceso a los archivos independientemente de su localización. VFS es una interfaz que proporciona el sistema operativo Linux para el acceso a los sistemas de archivos, de ahí que este sistema operativo pueda manejar simultáneamente tantos sistemas de archivo distintos.

Cuando realizamos una llamada al sistema de archivos, bien sea desde una aplicación o desde la línea de comandos, la interfaz que utilizamos es la que nos suministra el VFS. El VFS recibe la llamada, mira cual es el tipo de sistema de archivos al que va dirigida y le pasa la llamada, pero con el formato o interfaz del sistema de archivos de destino.

El VFS nos aporta independencia respecto al tipo de sistema de archivos, proporcionando una interfaz uniforme para el acceso a los archivos desde la capa de aplicación.

8.3. PROTOCOLO DE LLAMADA A PROCEDIMIENTO REMOTO

El protocolo de llamadas a procedimientos remotos, desarrollado por Sun Microsystems, abstrae la comunicación como procedimientos de servidor que pueden ser invocados desde el cliente. En RPC cada servicio está identificado por un número de programa y de versión. Además, a cada programa se le asigna un puerto.

Este protocolo es usado por varios de los sistemas de archivos distribuidos que veremos a continuación, por lo que se va a explicar con un poco de detalle. La arquitectura de un servicio cuando funciona sobre RPC se puede observar en la figura 8.4. Como podemos observar en la figura, desde el punto de vista de la arquitectura, el protocolo RPC se encuentra entre los niveles de transporte y aplicación.

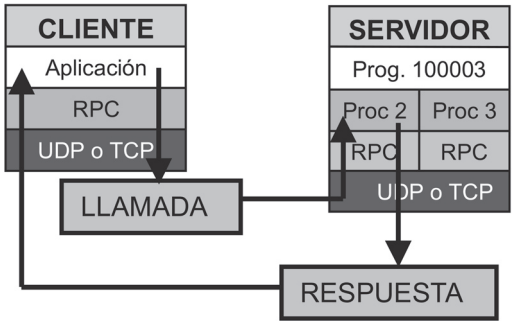


Figura 8.4. Arquitectura del protocolo RPC

Cuando una aplicación realiza una llamada a procedimiento remoto, desde el punto de vista de la aplicación no existe diferencia con respecto a una llamada a cualquier procedimiento o función local. La llamada realizada por la aplicación pasará al protocolo RPC, que se encargará de resolver toda la problemática relacionada con la comunicación entre nodos.

Al recibir la llamada desde la aplicación, el protocolo RPC transforma la llamada local en una llamada remota, insertando como parámetros el número de programa, versión y número de puerto dinámico asociado al programa. Después de formatear la llamada remota, se encapsula en un mensaje y se envía al servidor.

Cuando el servidor recibe el mensaje, se pasa al protocolo RPC del servidor, el protocolo extrae del mensaje la llamada a procedimiento e invoca

directamente al procedimiento del programa correspondiente. El protocolo RPC sabe que programa tiene que invocar por el número de puerto por el cual le ha llegado la petición, ya que *portmap* mantiene en todo momento los programas asociados a números de puerto dinámicos.

El resultado del procedimiento se pasará de nuevo al protocolo RPC, que lo encapsulará en un mensaje y lo devolverá al cliente. La capa RPC del cliente, recogerá el mensaje, obtendrá los datos de respuesta y se los pasará a la aplicación que solicitó la llamada inicial. Para la aplicación, el proceso es transparente, no siendo consciente de que la llamada se ha resuelto en un nodo remoto.

El funcionamiento del protocolo en la parte del cliente, descrito en los párrafos anteriores, se basa fundamentalmente en la utilización de los siguientes componentes:

- ID transacción.
- Versión RPC.
- Número de programa.
- Número de versión.
- Número de procedimiento.
- Parámetros del procedimiento.

Por otra parte, el mensaje de respuesta del servidor después de ejecutar el procedimiento pedido consta de los siguientes campos:

- ID transacción.
- Estado de salida.
- Resultados del procedimiento.

Otro de los aspectos que caracteriza al protocolo RPC es el uso de un modelo estándar de representación de datos externos. XDR (eXternal Data Representation, RFC 1014) es el mecanismo de representación utilizado para codificar los valores en las llamadas y respuestas RPC.

La característica principal de XDR es que representa los datos de manera independiente al formato de representación de cada máquina, por lo que permite la transferencia de datos entre máquinas de distintas arquitecturas y sistemas operativos. Entre otras cosas, define como se transmiten cada uno de los datos (orden de los bits y bytes, etc), incluye reglas para codificar estructuras XDR y un lenguaje de definición de estructuras de datos.

XDR permite la descripción y representación de los datos, para la descripción utiliza un lenguaje muy similar al lenguaje C. En cuanto a la representación, todos los elementos deben tener un tamaño que sea múltiplo de cuatro bytes, si algún tipo tiene longitud variable se añaden bytes a 0 para que sea múltiplo de cuatro. XDR soporta los siguientes tipos de datos y tamaños:

- Entero. 32 bits y notación en complemento a 2.
- Entero sin signo. 32 bits.
- Enumerado.
- Lógico.
- Hiper-enteros. Números de 64 bits con o sin signo.
- Punto flotante. Estándar de IEEE para números en punto flotante de precisión simple (32 bits).
- Punto flotante doble precisión. Estándar IEEE para números en punto flotante de doble precisión, concretamente 64 bits.
- Punto flotante de cuádruple precisión. Estándar IEEE para números de precisión extendida, representados con 128 bits.
- Datos sin interpretar. Datos sin tipo específico de longitud tanto fija como variable.
- Cadena. Cadena de bytes ASCII de longitud fija si se especifica el número de bytes, o máxima si no se define ningún tamaño.
- Tabla. Tipo de datos tabla de longitud fija o variable.
- Estructuras. Para codificar estructuras formadas por distintos tipos.
- Void. No especifica ningún tipo.
- Datos opcionales. Se utilizan para describir listas o árboles.

8.4. PROTOCOLO DE BLOQUE DE MENSAJE DEL SERVIDOR

El protocolo de bloque de mensaje del servidor o Server Message Block (SMB) es un protocolo de tipo petición-respuesta desarrollado inicialmente por IBM y modificado posteriormente por Microsoft para compartir recursos entre los nodos de una red. Éste último añadió nuevas características a la versión original como soporte para enlaces simbólicos, enlaces duros y mayor tamaño de archivos. La versión extendida de Microsoft se conoce como Common Internet File System o CIFS.

SMB ofrece principalmente los siguientes servicios:

- Acceso a archivos y directorios de otras máquinas de la red.
- Impresiones remotas. Mediante SMB se puede imprimir en impresoras que no estén conectadas físicamente a la máquina local.

- Autenticación mediante dominios Windows. En un dominio Windows podemos autenticarnos contra el servidor de dominio.
- Servicio de resolución de nombres WINS (Windows Internet Name Server). Para no tener que realizar broadcast o multicast, podemos configurar un nodo como servidor WINS, de manera que el resto de máquinas para resolver un nombre le pregunten exclusivamente al servidor WINS.

Debemos tener en cuenta algunos conceptos del mundo Windows. Se sigue la nomenclatura UNC (Universal Naming Convention), que antepone al nombre de la máquina dos barras invertidas y separa los directorios mediante otra barra invertida, un ejemplo es \\máquina\directorio. Además, debemos saber que la identificación de las máquinas se lleva a cabo mediante nombres de 15 caracteres como máximo, y no con números de cuatro octetos como estamos acostumbrados en TCP/IP.

Un concepto muy usado en entornos Windows es el de grupo de trabajo. Un grupo de trabajo es una etiqueta de nombre de grupo que identifica a un conjunto de máquinas y sus recursos sobre una red SMB. Pueden existir varios grupos de trabajo en una red al mismo tiempo. Otro aspecto al que debemos prestar atención es el concepto de dominio, que es lo mismo que un grupo de trabajo pero donde existe una máquina que ejerce las funciones de servidor, éste servidor es conocido como Controlador Principal de Dominio. Cuando un usuario accede al sistema, puede registrarse en la máquina local o en el dominio a través del servidor.

Para comprender mejor la arquitectura sobre la que se sustenta tanto el protocolo SMB como Samba es conveniente hacer un poco de historia.

Originalmente, IBM diseño un protocolo para comunicar sus computadoras en red llamado Network Basic Input/Output System o NetBIOS, éste permitía a las aplicaciones conectarse y compartir datos con otros nodos. Posteriormente, se extendió la funcionalidad añadiendo un protocolo de transporte y pasó a llamarse NetBIOS Extended User Interface o NetBEUI. NetBEUI era el protocolo sobre el que funcionaba SMB en las primeras versiones de Windows. NetBIOS identificaba las máquinas mediante nombres de 15 caracteres como máximo y como Microsoft lo adoptó como mecanismo de comunicación para sus sistemas, éstos se identifican de igual manera.

Pero NetBIOS se ha implementado sobre otros protocolos, por ejemplo, IPX de Novell o TCP/IP. Las implementaciones de Windows más actuales están basadas en estos documentos, así como el paquete de utilidades Samba que analizaremos a continuación.

8.5. CASO DE USO 1: SISTEMA DE ARCHIVOS DE RED

El sistema de archivos en red o Network File System (NFS) permite acceder a los archivos remotos exactamente igual que si fueran locales, siendo el acceso a la información transparente a la aplicación cliente. NFS constituye la implementación en el mundo UNIX de un sistema de archivos remoto.

8.5.1. Arquitectura

La arquitectura del sistema de archivos NFS sigue el modelo cliente-servidor. En la parte del cliente, la funcionalidad está soportada por el núcleo del sistema operativo y es éste el responsable del protocolo y la transferencia de la información. Por el contrario, en el lado del servidor, la funcionalidad no está implementada en el sistema operativo, por lo que la llevará a cabo el demonio o servicio *nfsd* en el nivel de aplicación.

En la arquitectura se debe resaltar que NFS no funciona directamente sobre TCP/IP, sino que lo hace a través del protocolo de llamadas a procedimientos remotos o RPC (Remote Procedure Calls). Dentro de los servicios que funcionan sobre RPC, NFS es el servicio más extendido, siendo utilizado en multitud de ambientes y escenarios (fugura 8.5).

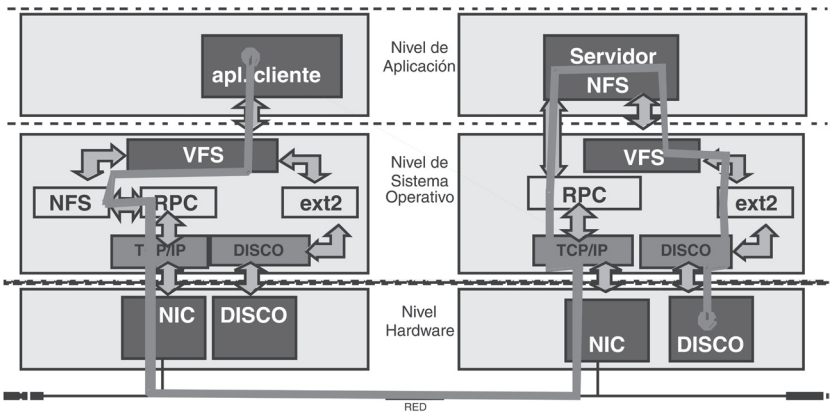


Figura 8.5. Arquitectura del sistema de archivos de red NFS

8.5.2. Servicios

Se ha comentado anteriormente que, si bien en el cliente el sistema de archivos distribuido tiene que estar soportado por el núcleo del sistema operativo, en el servidor es la capa de aplicación la encargada de dar soporte al sistema de archivos de red NFS.

El nivel de aplicación utiliza tres servicios o demonios para soportar el sistema NFS, es decir, en el lado del servidor necesita de la ejecución de tres demonios para poder ofrecer el servicio NFS, estos tres servicios son *portmap*, *mountd* y *nfsd*.

8.5.2.1. Demonio portmap

Los servicios que se ejecutan sobre el protocolo RPC no tienen un número de puerto privilegiado asociado, sino que se asigna un número de puerto en el momento de ejecutarlos. Portmap es un servicio que se encarga de asociar el número de programa RPC al puerto TCP/UDP donde se encuentra escuchando el demonio correspondiente. Los programas que usan RPC se registran en portmap para poder ser accesibles.

El servidor portmap *(/sbin/portmap)* sí tiene un número de puerto definido en el cual escucha, el número 111. Es lógico que portmap tenga un número de puerto definido y perfectamente conocido, ya que los clientes remotos le solicitarán información sobre el número de puerto de otros programas o servicios y deben saber exactamente el número de puerto en el que escucha el demonio portmap.

Los programas que soliciten servicios de un demonio que funcione sobre un puerto dinámico serán informados por portmap, pero ¿cómo puede el administrador del sistema saber que demonios se están ejecutando en un momento dado sobre puertos dinámicos?, mediante la orden *rpcinfo –p* se puede conocer en todo momento la tabla de asociación entre los demonios que están funcionando sobre RPC y los puertos en los que están escuchando.

El listado 8.1 muestra el resultado de la ejecución del comando anterior, en la información mostrada podemos observar el número de programa, su versión, el protocolo de la capa de transporte por el que funciona el servicio, el puerto asociado y el nombre del programa. Podemos ver que algunos servicios funcionar indistintamente por tcp o udp, este es el caso de NFS, que por defecto funciona por udp, pero se puede configurar para que funcione de forma confiable sobre tcp.

```
#rpcinfo -p
Programa vers proto   Puerto
  100000    2   tcp    111  portmapper
  100000    2   udp    111  portmapper
  100003    2   udp   2049  nfs
  100003    3   udp   2049  nfs
  100003    2   tcp   2049  nfs
  100003    3   tcp   2049  nfs
  100021    1   udp  32768  nlockmgr
  100021    3   udp  32768  nlockmgr
  100021    4   udp  32768  nlockmgr
  100021    1   tcp  32769  nlockmgr
  100021    3   tcp  32769  nlockmgr
  100021    4   tcp  32769  nlockmgr
  100005    1   udp    862  mountd
  100005    1   tcp    865  mountd
  100005    2   udp    862  mountd
  100005    2   tcp    865  mountd
  100005    3   udp    862  mountd
  100005    3   tcp    865  mountd
```

Listado 8.1. Ejecución de la orden rpcinfo

8.5.2.2. Demonio mountd

Cuando un cliente intenta montar un volumen remoto mediante NFS se lo solicita al servidor mountd. Este servicio es el programa RPC número 100005 y tiene varias versiones (listado 8.1).

Si el sistema de archivos se puede montar por el equipo cliente devuelve un descriptor de volumen, que será utilizado posteriormente para acceder al sistema de archivos remoto como si del sistema local se tratase. La figura 8.6 muestra el acceso al punto de montaje /usr en el nodo cliente, y su enlace real a la ruta /usr del nodo servidor.

Es probable que el volumen no se pueda montar debido a temas de permisos, se verá este aspecto posteriormente.

Como en el servicio anterior, se dispone de una orden para saber los directorios montados en un momento determinado y el nombre del ordenador que tiene montados cada uno de los directorios. La orden *showmount –a* muestra esta información.

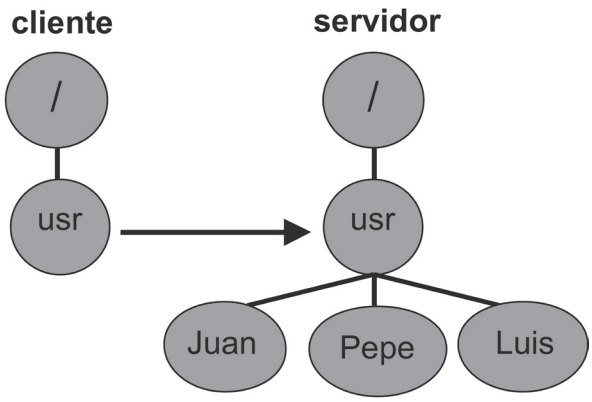


Figura 8.6. Montaje de un directorio remoto

8.5.2.3. El servicio nfsd

De todos los servicios involucrados en el sistema de archivos remoto NFS, el demonio *nfsd* constituye el más importante. Si los servicios anteriores son servicios de apoyo, uno de registro de puertos y otro de montaje del directorio remoto, nfsd representa el servicio principal que se encarga de la gestión del sistema de archivos, transferencia de la información, etc.

Utiliza el descriptor devuelto por el demonio *mountd*, descriptor que también es usado por los clientes en sus solicitudes. Las aplicaciones cliente realizarán peticiones comunes de acceso a archivos como si de un archivo local se tratase, la petición se envía al sistema remoto donde está el servidor *nfsd*, atiende la petición y transfiere la información.

El demonio nfsd es la aplicación RPC número 100003, existen las versiones 2 y 3, y aunque como cualquier otra aplicación RPC no tiene un número de puerto conocido asociado, la mayoría de implementaciones utilizan el puerto 2049.

La figura 8.7 muestra la secuencia de pasos necesaria para el funcionamiento del servicio de archivos de red.

La descripción de cada uno de los pasos se explica a continuación:

1. El demonio portmap debe estar ejecutándose en el servidor, podemos comprobar si está en ejecución mediante la orden *ps –aux | grep portmap*.

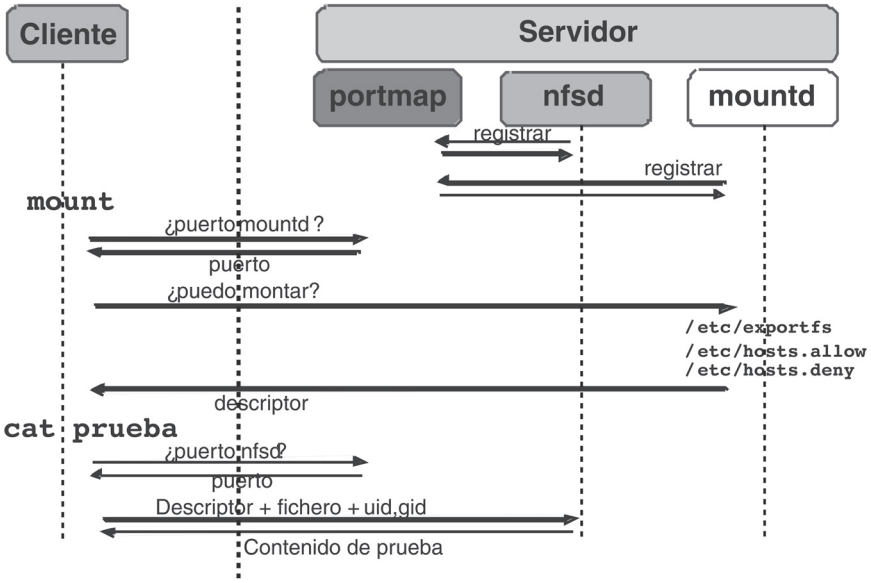


Figura 8.7. Esquema de funcionamiento NFS

2. Cuando se arranca el servicio nfsd se registra en portmap, o lo que es lo mismo solicita un número de puerto dinámico. Portmap le asignará el número de puerto y lo registrará en la tabla de asignación de puertos.
3. Igual que el demonio anterior, cuando se lanza mountd se registra o le solicita número de puerto a portmap, que le asignará el puerto y lo registrará en la tabla. En este momento, el servidor está preparado para aceptar peticiones de montaje de sistema de archivos remoto NFS.
4. El cliente intenta montar un sistema de archivos remoto NFS en algún punto de montaje local invocando el comando local *mount*. Al comando mount se le pasará el tipo de sistema de archivos mediante la opción *–t*, o bien lo deducirá si el dispositivo de montaje empieza con `nombre_maquina:path`, donde nombre_maquina es el nombre del servidor. El comando mount local solicitará al demonio portmap del servidor el número de puerto en el que está escuchando el demonio mountd en el servidor.

5. Después de obtener el número de puerto de mountd en el servidor, el comando mount cliente solicitará a dicho demonio el montaje del sistema de archivos remoto. Si el montaje se puede llevar a cabo, el demonio mountd montará el sistema y devolverá un descriptor al cliente, a partir de este momento el cliente puede acceder al sistema de archivos remoto de forma transparente. El hecho de que se pueda montar o no, depende de aspectos tales como la configuración y la seguridad, por lo que lo abordaremos con detalle posteriormente en el apartado de administración.
6. Supongamos que accedemos al sistema de archivos remoto para mostrar el contenido de un archivo con el comando *cat*. El comando cat solicitará al demonio portmap del servidor el número de puerto en el que está escuchando el demonio nfsd en el servidor.
7. Después de obtener el número de puerto de nfsd en el servidor, el comando cat del cliente solicitará la información del archivo remoto a través del número de puerto obtenido, y utilizando el descriptor facilitado anteriormente por el servicio mountd.

En la descripción anterior del proceso, la petición de puerto la hacen los comandos por simplicidad en la exposición. No obstante, es importante notar que cuando el cliente ejecuta los comandos de montaje o visualización de archivos, éstos son enviados al VFS del nodo cliente. El VFS deducirá que se trata de un sistema de archivos de red y pasará las peticiones al módulo NFS del núcleo. Será el módulo NFS del núcleo el encargado de solicitar el número de puerto dinámico al demonio portmap del servidor y no los comandos o aplicaciones, para los cuales el proceso es transparente.

8.5.3. Instalación

En la instalación del servicio de archivos de red se debe distinguir entre el nodo cliente y el servidor del servicio, ya que como se comentó anteriormente en el cliente el responsable de soporte del protocolo NFS es el núcleo del sistema operativo, mientras que en el servidor la responsabilidad recae en los servicios estudiados en el apartado anterior.

Antes de utilizar NFS debemos estar seguros que nuestro núcleo soporta el tipo de sistema de archivos, para comprobar si el núcleo del sistema soporta nfs podemos visualizar el archivo *filesystems* del directorio virtual */proc* (listado 8.2). En el archivo referenciado se relacionan, uno en cada línea, los distintos tipos de sistemas de archivos soportados por el núcleo.

```
#cat /proc/filesystems
nodev rootfs
nodev bdev
nodev proc
nodev sockfs
nodev tmpfs
nodev shm
nodev pipefs
      ext2
nodev ramfs
      vfat
      iso9660
nodev devpts
      ext3
nodev usbdevfs
nodev usbfs
nodev autofs
nodev nfs
```

Listado 8.2. Comprobación del soporte NFS del núcleo

En el caso de que no aparezca una línea con NFS, debemos compilar el núcleo de nuevo con soporte para el sistema de archivos de red. En el apartado de sistema de archivos, el script de configuración nos pregunta si deseamos soporte para NFS (*NFS filesystem support (CONFIG NFS FS) [y]*), a lo que contestaremos [y] indicando que sí. De esta manera, después de compilar y cargar la nueva imagen, tendremos habilitado el soporte en el núcleo del sistema de archivos de red.

Desde el punto de vista del servidor, y como hemos analizado anteriormente, la instalación consiste en tener disponibles y ejecutándose los demonios *mountd* y *nfsd*. En caso de no disponer de tales servicios, podemos obtener en www.sourceforge.net el paquete *nfs-utils* que añadirá los citados servicios, permitiendo que nuestro sistema pueda usarse como servidor NFS.

8.5.4. Administración

La administración como en la mayoría de servicios de red se realiza editando archivos de configuración tanto del servicio como del sistema.

En el cliente puede no haber ningún tipo de administración o configuración y limitarnos simplemente a ejecutar la orden *mount* cuando queramos montar un sistema de archivos de red, pero debemos tener en cuenta que la orden se ejecuta sólo con el usuario *root*. El listado 8.3 muestra dos formas distintas de ejecutar el comando, la primera indicando el tipo de sistema de archivos y la segunda deduciéndolo el sistema por encontrar el nombre de nodo *asi01*.

No parece lógico que un administrador tenga que montar las rutas exportadas por un servidor de forma manual, debemos realizar el proceso de forma automatizada. Para ello lo que se hace es montarlo en el arranque del sistema, como si fuera un sistema de archivos local más. La forma que tiene el administrador de cargar sistemas de archivo en el arranque es editando la tabla de sistemas de archivo, o más concretamente el fichero */etc/fstab*. Por lo tanto, para que monte un sistema NFS en el arranque de la máquina debemos añadir una entrada en este archivo.

El listado 8.3 muestra un ejemplo de archivo */etc/fstab* con tres ejemplos de montaje en el arranque de sistemas remotos NFS. En el ejemplo se pueden observar cuatro columnas: sistema de archivo remoto que se pretende montar, punto de enlace en el sistema local, tipo de sistema de archivos y opciones. Abordaremos las distintas opciones posteriormente en este mismo apartado.

```
#mount –t nfs asi01:/home /home

#mount asi01:/home /home

#cat /etc/fstab

Asi01:/home        /home     nfs soft
192.168.2.56:/home /home/tmp nfs rsize=2048,wsize=2048
192.168.6.57:/etc  /home/etc nfs nolock
```

Listado 8.3. Tabla de sistema de archivos para NFS.

Para administrar el sistema de archivos de red en el servidor, o lo que es lo mismo, para convertir nuestro sistema en un servidor NFS tenemos que editar cuatro archivos de configuración: /etc/exports, /etc/hosts, /etc/host.allow, /etc/host.deny. Sólo es estrictamente necesario editar el primero de los archivos, aunque analizaremos la función de cada uno de ellos.

El archivo principal de administración del servidor NFS es /etc/exports, donde indicamos que rutas o directorios del sistema local exportamos o compartimos. El listado 8.4 muestra un ejemplo del archivo de configuración, donde se exportan cuatro rutas distintas. Cada entrada en el archivo consta de la ruta a exportar, el nombre de la máquina a la que se le permitirá montar la ruta y las opciones de montaje. Como podemos observar, se puede permitir montar a distintas máquinas y con distintas opciones, pero debemos tener en cuenta que damos permisos a máquinas y no a usuarios.

Es importante indicar que si modificamos el archivo */etc/exports* cuando los servidores se encuentran en ejecución, no necesitamos parar y reiniciar los demonios para que vuelva a cargar la nueva configuración. Para que se activen los cambios, sólo necesitamos ejecutar la orden *exportfs –a*, este comando obligará al demonio *nfsd* a leer de nuevo el archivo de configuración sin necesidad de detenerlo.

```
#cat /etc/exports

 /               master(rw) trusty(rw,no_root_squash)
 /projects       proj*.local.domain(rw)
 /usr            *.local.domain(ro) @trusted(rw)
 /home/joe       pc001(rw,all_squash,anonuid=150,anongid=100)
```

Listado 8.4. Archivo principal de configuración NFS.

Los archivos /etc/host.allow y /etc/host.deny son utilizados por el demonio mountd para comprobar si se les permite o deniega respectivamente el acceso a los clientes. Las entradas en los dos archivos tienen el mismo formato: nombre del servicio que se ofrece, máquina o máquinas y si se permite o deniega (ALLOW o DENY).

El servidor mountd usa el archivo /etc/hosts para resolver la dirección IP del nodo cliente, en caso de que no se use DNS. En caso de usar DNS, se obtendrá el nombre canónico u oficial (una máquina puede tener varios nombre, pero sólo uno oficial), por lo que no debemos utilizar alias en el archivo principal de configuración /etc/exports.

Anteriormente, al describir el funcionamiento del servicio NFS, decíamos que el demonio mountd podría montar o no el sistema de archivos dependiendo de aspectos tales como la configuración y la seguridad. Analizaremos ahora el funcionamiento de la orden de montaje, explicando la utilización y validación de los archivos de configuración.

Supongamos el archivo */etc/exports* del listado 8.4 y supongamos que una máquina cliente llamada *asi01* solicita montar remotamente el directorio raíz. El demonio mountd recibirá la petición, mirará el archivo en busca de una entrada con el directorio raíz y después mirará si *asi01* es uno de los nodos a los que se exporta el directorio. En este caso el directorio raíz no se exporta para el citado nodo, por lo que no se montará el sistema de archivos y se denegará el acceso.

Supongamos que el nodo *master* solicita montar también el directorio raíz. El demonio mountd sigue la misma secuencia del párrafo anterior, en este caso el directorio raíz sí se exporta para el citado nodo. Después de comprobar que se permite exportar el directorio raíz al nodo *master*, el demonio *mountd* mirará el archivo */etc/host.allow* en busca de una entrada que permita al nodo *master* acceder al servicio *nfsd*, en cuyo caso montará el sistema de archivos y devolverá el descriptor correspondiente. Si no existe la citada entrada en /etc/host.allow, entonces mirará el archivo */etc/host.deny* en busca de una entrada que deniegue al nodo *master* acceder al servicio *nfsd,* en cuyo caso no montará el sistema de archivos y denegará el acceso. Finalmente, si no existe entrada en ninguno de los dos archivos, permitirá el acceso y montará el directorio raíz.

En los archivos de configuración se puede observar el uso de determinadas opciones tanto en el cliente como en el servidor. La tabla 8.1 muestra las opciones disponibles en el cliente, mientras que la tabla 8.2 muestra las opciones disponibles en el servidor.

OPCIÓN	DESCRIPCIÓN
rsize=n y wsize=n	Especifican el tamaño de datagrama utilizado por el cliente NFS en las peticiones de lectura y escritura, respectivamente. Por defecto, cada una de ellas vale 1024 octetos, dados los límites del tamaño de datagrama UDP.
Timeo=n	Esta opción establece el tiempo máximo de espera de respuesta a una petición del cliente NFS; en centésimas de segundo. Por defecto, este valor es de 0.7 segundos.
Hard	Marca el montaje del volumen como físico. Es un valor por defecto.
Sofá	Hace que el montaje sea solo lógico (opuesto al anterior).
Intr.	Esta opción habilita la posibilidad de que una señal interrumpa una espera por NFS.

Tabla 8.1. Opciones de montaje del cliente NFS.

OPCIÓN	DESCRIPCIÓN
insecure	Permitir acceso no autentificado desde ese nodo.
unix-rpc	Requerir autenticación RPC del dominio Unix para este nodo. Se trata simplemente de que las peticiones se originen en un puerto reservado (es decir, inferior al 1024). Esta opción está activa por defecto.
Secure-rpc	Requerir autenticación RPC segura para este nodo. Aun no está implementado. Se sugiere ver la documentación de Sun al respecto (véase, "Secure RPC").
Kerberos	Requerir autenticación Kerberos. Tampoco se ha implementado aun. Se sugiere consultar la documentación del MIT.
root_squash	Se trata de una opción de seguridad que deniega acceso a nivel de superusuario, traduciendo el identificador uid recibido (0) al del usuario nobody. Es decir, cualquier petición NFS del usuario root será tomada como si fuera del usuario nobody.
no_root_squash	Evita la restricción anterior. Es una opción por defecto.
Ro	Monta la jerarquía de ficheros en modo de solo lectura. Es una opción por defecto.
Rw	Monta el directorio con permiso para leer y escribir en él.
link_relative	Convierte enlaces simbólicos absolutos enlaces relativos colocando los prefijos "../ " que sean necesarios para hacer que apunten a la raíz del servidor. Esta opción solo tiene sentido cuando se monta un sistema de ficheros completo y no solo un directorio. Es una opción activa por defecto.
link_absolute	Deja los enlaces absolutos como estaban (es la opción habitual en servidores NFS de Sun).
map_identity	La opción map_identity indica al servidor que asuma que el cliente utiliza el mismo mapa de uids y gids que el servidor. Es una opción por defecto.
map_daemon	Esta opción indica al servidor NFS que no comparte el mapa de usuarios con el del cliente. Con ello, las comparaciones de uids y gids se harán mediante una lista de mapeado entre ambos que se construirá llamando al demonio ugidd del cliente.

Tabla 8.2. Opciones de montaje del servidor NFS.

8.5.5. Usuarios

Es muy importante resaltar que no existe ningún proceso de acreditación de usuarios en NFS, por lo que el administrador debe decidir con cautela a qué ordenadores exporta un determinado directorio. Un directorio sin restricciones es exportado, en principio, a cualquier otro ordenador conectado con el servidor a través de la red. Si en un ordenador cliente NFS existe un usuario con un UID igual a "X", este usuario accederá al servidor NFS con los permisos del usuario con el UID igual a "X" del servidor, aunque se trate de usuarios distintos.

Es posible, por otra parte, que el mismo usuario tanto en el cliente como en el servidor no pueda acceder vía NFS a sus datos en el servidor desde el cliente, por tener los usuarios distintos UID.

Por lo tanto existen problemas en la gestión de usuarios entre el cliente y el servidor cuando utilizamos NFS. Existen opciones en el servidor (tabla 8.2) que nos permiten tener cierto control del proceso; root_squash, no_root_squash, map_identity y map_daemon, pero no alcanzan la profundidad o flexibilidad deseadas. El número de identificación de usuario debe ser igual en todos los sistemas desde los que se pueda conectar.

Hay que unificar los mapas de usuario del cliente y del servidor, y para ello lo recomendable es utilizar los sistemas de autenticación estándar más usados en la realidad.

- Network Information Service o NIS. La opción más sencilla y usada para centralizar la gestión de cuentas y el acceso a archivos remotos es utilizar NIS y NFS. Los dos sistemas se complementan perfectamente, no obstante, los dos fueron creados por Sun Microsystems y utilizan el subsistema RPC como soporte. Con el uso de NIS tendremos las cuentas centralizadas en un único mapa de usuarios, por lo que no tendremos problemas con los UID. Realmente un sistema NIS es una base de datos distribuida, pero es el servidor el único que puede modificar y gestionar los datos o mapa de usuarios, distribuyendo copias a los clientes.
- Lightweight Directory Access Protocol o LDAP. El protocolo de acceso a directorio es una base de datos optimizada para lecturas, que nos permite, entre otras cosas, tener las cuentas de usuario y autenticarnos. Mediante el uso conjunto de NFS y LDAP podemos tener un único mapa de usuarios y evitar los problemas relacionados con éstos en el acceso a los archivos remotos. El libro trata de forma extensa el protocolo LDAP en otro tema.

8.6. CASO DE USO 2: SAMBA

En el presente apartado vamos a analizar el paquete de utilidades *Samba*, que es la implementación Unix del protocolo SMB (Server Message Block). Mediante Samba podremos tener redes formadas por máquinas Windows y Linux o Unix indistintamente. La figura 8.8 muestra la visión de una máquina Linux desde Windows, y como se accede a ella de manera análoga a como se accede a cualquier otro ordenador en una red Windows.

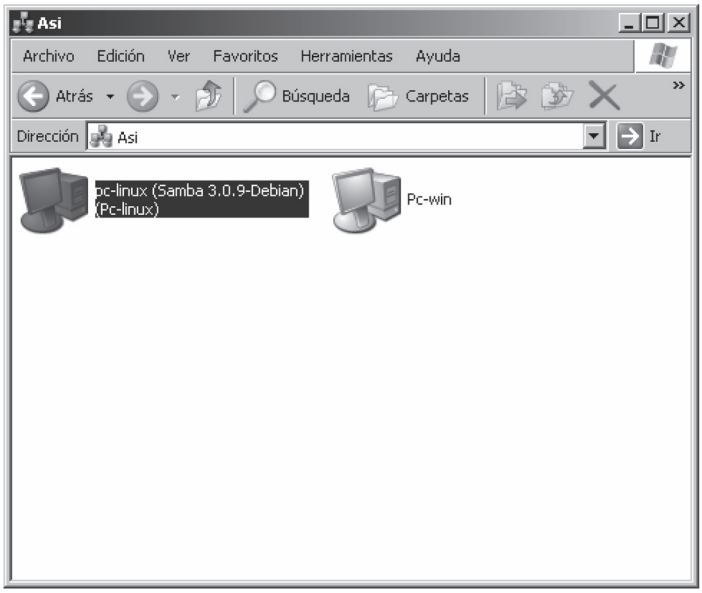


Figura 8.8. Vista de una máquina Linux desde Windows.

El paquete de utilidades Samba nos permite ofrecer todas las funciones del protocolo SMB, pero desde una máquina Linux o Unix. Podemos compartir nuestro disco con otras máquinas Windows, permitir que máquinas Windows impriman en nuestras impresoras, convertir una máquina Linux en un servidor de autenticación de un dominio Windows y, por supuesto, nuestro nodo Linux o Unix se puede comportar como un servidor WINS.

Aunque vamos a analizar las utilidades con detalle más adelante, el paquete de utilidades Samba está formado por los siguientes componentes:

- smbd. Es el servidor principal, se encarga de gestionar los recursos compartidos por la máquina servidora en la que está Samba y atender las peticiones de los clientes. Realiza todas las funciones del párrafo anterior excepto la de servidor WINS.
- nmbd. Demonio que ofrece la funcionalidad de servidor de nombres WINS y resolución de nombres NetBIOS.
- smbclient. Cliente que funciona como un FTP y que se puede usar para conectarnos a recursos compartidos por Samba.
- smbtar. Utilidad similar al comando *tar* para realizar copias de seguridad de datos situados en los recursos compartidos.
- nmblookup. Programa que nos permite realizar búsquedas de nombres NetBIOS sobre TCP/IP.
- nmbpasswd. Herramienta para cambiar las contraseñas cifradas utilizadas por Samba.
- smbstatus. Nos permite conocer en todo momento el estado de las conexiones actuales a recursos compartidos por nuestro servidor Samba.
- testparm. Nos ofrece la posibilidad de testear el archivo de configuración *smb.conf*.

8.6.1. Instalación

Como en la mayoría de paquetes de software, la instalación dependerá de si obtenemos los archivos fuentes de la distribución o los binarios. Obviamente, la forma más sencilla es descargarnos los binarios, descomprimirlos y copiarlos en los directorios por defecto.

Pero, en este caso, vamos a abordar la instalación del paquete desde el código fuente, debido a que la última versión no suele estar disponible para todas las plataformas.

La compilación e instalación del paquete Samba consta de los pasos que se detallan a continuación:

- Conectarse a la dirección http://www.samba.org y descargarse los fuentes de la última distribución. En el momento de escribir este libro, mayo de 2007, la última versión estable del paquete es Samba-3.0.25.

- Leer la documentación sobre la instalación. La documentación suele traer notas sobre la instalación en las distintas plataformas, así como distintos problemas que nos podemos encontrar en el proceso de instalación.
- Configurar la compilación. La configuración permitirá adaptar Samba a nuestro sistema operativo y con las variables globales que le indiquemos. Para configurar Samba tenemos que ejecutar el programa *configure*, situado en el directorio *samba-3.0.25/source*. Podemos indicarle opciones globales mediante la opción *--with*, por ejemplo, *./configure --with-ssl* configurará Samba con soporte para SSL. Existen muchas opciones, como soporte para Kerberos o LDAP, las distintas opciones que soporta el programa las podemos conocer con el comando *./configure--help*.
- Compilar los fuentes. Si todo el proceso de configuración se ha resuelto con éxito, estamos en disposición de compilar los fuentes de la distribución y generar los ejecutables de Samba. Para ello, ejecutamos el comando *make* desde el mismo directorio en el que nos encontrábamos.
- Después de la compilación, sólo nos queda instalar los ejecutables en los directorios seleccionados en la configuración, en la mayoría de distribuciones el directorio base es */usr/local/samba*. Ésta operación la llevaremos a cabo con la orden *make install*.

Hemos terminado de compilar e instalar nuestra distribución Samba, y estamos preparados para lanzar sus servicios, sólo falta crear un fichero de configuración, aspecto que abordaremos en el apartado de administración. En el siguiente apartado vamos a analizar los dos servicios principales que nos aporta el paquete Samba.

8.6.2. Servicios

El funcionamiento del paquete Samba se basa en el uso de dos servicios y un conjunto de utilidades. El protocolo SMB está implementado en los dos servicios principales: *smbd* y *nmbd*. Además, disponemos de un conjunto de utilidades para testear el servidor, conectarnos como clientes a recursos SMB o identificar las conexiones activas a recursos, entre otras. En este apartado vamos a analizar los procesos servidores que proporcionan la funcionalidad del protocolo y en un apartado posterior abordaremos algunas de las utilidades.

Antes de analizar la utilidad de cada uno de los servicios, veamos las distintas formas que tenemos de ejecutarlos. Asumiendo que Samba se ha instalado en */usr/local/samba* (opción por defecto), podemos lanzar los servicios manualmente mediante la ejecución de los comandos siguientes:

```
/usr/local/samba/bin/smbd -D
/usr/local/samba/bin/nmbd -D
```

Otra opción de arranque de los demonios es automáticamente en el arranque del sistema, para ello existe un script llamado *smb* que ejecuta los dos servicios. Tendremos que enlazar el script en el directorio de arranque *rcX.d* correspondiente.

Finalmente, la última opción de arranque de los demonios es a través de *inetd*. Éste demonio escucha los puertos indicados en el archivo */etc/services* y lanza el servicio correspondiente en el caso de que lleguen peticiones. Tiene la ventaja de no sobrecargar el sistema, ya que los servicios no están activos hasta que lleguen peticiones y pueden no llegar nunca. Presenta el inconveniente de que la primera petición de servicio tarda más en resolverse, debido a que primero se debe lanzar el servicio correspondiente.

Para configurar el arranque de los servicios de Samba a través del demonio de red *inetd*, debemos editar dos archivos de configuración. El primer archivo es el mencionado */etc/services*, donde le indicamos los puertos que debe escuchar inetd y los servicios que tiene que lanzar. Tendremos que añadir las entradas siguientes:

```
netbios-ssn 139/tcp
netbios-ns 137/udp
```

El segundo y último archivo a editar es */etc/inetd.conf*, donde indicamos el demonio a ejecutar y sus opciones. Debemos añadir las dos entradas siguientes:

```
netbios-ssn stream tcp onwait root /usr/local/samba/bin/smbd
smbd
netbios-ns dgram udp wait root /usr/local/samba/bin/nmbd nmbd
```

8.6.2.1. Demonio smbd

Es el servicio o demonio principal de Samba y el que implementa la funcionalidad básica del protocolo SMB. Sobre smbd recaen las funciones principales siguientes:

- Compartir uno o más sistemas de archivos.
- Compartir impresoras, instaladas tanto en el servidor como en los clientes.
- Comportarse como visualizador de red para los clientes.
- Autenticar y autorizar a los clientes contra un dominio Windows.

La principal función de Samba es compartir archivos o unidades de disco. En este sentido, y siempre que esté bien configurado, podemos hacer que una máquina Linux o Unix se comporte como cualquier máquina Windows. Desde el explorador de Windows o utilizando nomenclatura UNC podemos acceder a los archivos o discos compartidos por un sistema Linux, exactamente igual que lo hacemos a una máquina Windows.

Pero, el servidor Samba no sólo permite compartir archivos o discos sino que también puede compartir impresoras. Cualquier cliente Windows podrá enviar los trabajos de impresión a la impresora compartida por el servidor Samba, los trabajos serán encolados en orden de llegada. Para usarla desde un cliente Windows, sólo tenemos que añadir una nueva impresora e indicarle (mediante el explorador o en notación UNC) que es una impresora de red, Windows instalará los manejadores y podremos imprimir en la impresora de red compartida por el servidor Samba.

Hemos comentado en la introducción de este apartado el concepto de dominio Windows, que es un grupo de trabajo pero con una máquina que realiza las funciones de controlador principal de dominio. El controlador de dominio es el componente central del dominio, de la misma forma que un servidor NIS lo es del servicio de información de una red Linux o Unix.

La función principal del controlador de dominio es la autenticación. La autenticación es el proceso de permitir o denegar a un usuario el acceso a los recursos compartidos o a otra máquina de la red, normalmente a través del uso de una contraseña. Para validar las contraseñas, el controlador de dominio utiliza un gestor de cuentas de seguridad (SAM, Security Account Manager) que mantiene una base de datos con las combinaciones usuario-contraseña correctas que permiten el acceso al sistema.

El enfoque centralizado de los controladores de dominio es más eficiente que mantener en cada máquina las contraseñas para cada recurso de red disponible, por el contrario, tiene la desventaja de que el servidor debe estar necesariamente funcionando para permitir la validación y el acceso a los recursos.

En un dominio Windows, cuando un cliente solicita acceso a los recursos compartidos de una máquina, ésta pregunta al controlador de dominio si el usuario está autenticado. Si lo está, la máquina establecerá una conexión con el cliente y utilizará la autorización de acceso para el recurso y usuario concretos. Si el usuario no está autenticado, la conexión se negará. Debemos tener en cuenta que, cuando un usuario se registra en el controlador de dominio, recibe una ficha de autenticación de manera que no necesitará registrarse en otros recursos del dominio.

Pues bien, el servidor o demonio *smbd* puede ejercer las funciones propias de un controlador de dominio en un dominio Windows. De esta forma, los usuarios pueden autenticarse contra el servidor *smbd* y, además, las máquinas que comparten recursos pueden preguntarle al demonio *smbd* si un usuario está o no autenticado.

Otro concepto del que no hemos hablado hasta ahora es el concepto de controlador de seguridad. El controlador de seguridad se encarga de replicar periódicamente la base de datos del controlador principal de dominio, de esta forma, en el caso de que el controlador principal caiga, el controlador de seguridad asumirá sus funciones, permitiendo la continuidad del servicio. Como habréis deducido, el demonio smbd puede ejercer las funciones de controlador de seguridad (sólo a partir de la versión 3), ahora bien, la documentación de Samba recomienda que se utilice como backend o base de datos LDAP.

Finalmente, la última función que cumple el servicio *smbd* es la de visualización o browsing. Cuando en un cliente de una red SMB queremos saber las máquinas que están accesibles en un momento dado, lo que se hace es preguntar a un servidor de visualización maestro. Es decir, existe una máquina que se encarga de mantener una lista de máquinas activas, el resto de máquinas en vez de mantener su propia lista le preguntan ésta, este enfoque reduce el tráfico de red. El demonio smbd puede funcionar como servidor de visualización maestro y mantener la lista de máquinas activas en un grupo de trabajo o dominio.

8.6.2.2. Demonio nmbd

El demonio *nmbd* es menos importante que el anterior, pero constituye junto a *smbd* el bloque principal del paquete de utilidades Samba. El servicio *nmbd* se encarga de imitar la funcionalidad de un servidor de Windows WINS (Windows Internet Name Server).

Existen dos funciones principales en un servidor WINS:

- Registro de nombres. Cuando se arraca una máquina NetBIOS, debe registrar su nombre en un servidor de nombres NetBIOS (NBNS, NetBIOS Name Server), para que el servidor compruebe que no existen nombres duplicados o resuelva el conflicto en el caso de duplicidades.
- Resolución de nombres. De la misma forma que un servidor DNS resuelve nombres DNS y devuelve direcciones IP, un servidor WINS debe devolvernos la dirección IP asociada a un nombre NetBIOS.

Las funciones anteriores se pueden llevar a cabo mediante broadcast, sin servidor de nombres. Cuando un nodo se activa puede enviar un paquete al resto de nodos indicando que quiere registrar su nombre, si nadie le contesta indicándole que el nombre ya está en la red, asumirá que el nombre no está registrado.

En cuanto a la resolución de nombres, se puede enviar un mensaje de broadcast al resto de máquinas solicitando la resolución de un nombre, la máquina que se sienta identificada devolverá al remitente su dirección IP. La ventaja del servidor de nombres es que reduce enormemente el tráfico generado en la red.

8.6.3. Administración

Como en la mayoría de servicios, la administración se lleva a cabo editando o modificando archivos de configuración. En el caso de samba el archivo de configuración se denomina *sbm.conf*, el listado 8.5 muestra un ejemplo típico. Samba tiene más de 200 opciones distintas de configuración, en nuestro caso sólo vamos a estudiar algunas de ellas, suficientes para poner el servicio en funcionamiento. El lector puede ampliar información en la documentación de Samba o en la Web oficial.

```
[global]
      netbios name = GAIA
      server string = Samba %v
      workgroup = ASI
      valid users = root, mora, paco, diego, virgilio, antonio

[printers]
      guest ok = yes
      printable = yes

 [temp]
      comment = Directorio temporal
      path = /home/temp
      read only = no
      guest ok = no
      valid users = paco, virgilio
```

Listado 8.5. Archivo de configuración sbm.conf

Lo primero que debemos resaltar del archivo de configuración de Samba es que está dividido en secciones, cada sección está asociada a un recurso o servicio y se indica mediante nombres encerrados entre corchetes. Todas las secciones del archivo de configuración, excepto la sección *global*, indican discos o impresoras compartidas para los clientes de la red SMB a la que pertenece el servidor.

Dentro de cada sección deben aparecen varias opciones, las que queramos que se apliquen al recurso o servicio correspondiente. Cada opción está identificada por el nombre de la directiva y su valor. Además, debemos tener en cuenta que las opciones de la sección *global* se aplican a todas las secciones. En este sentido, si una opción aparece en la sección global y posteriormente en otra sección con un valor distinto, prevalecerá el último valor.

En nuestro ejemplo, la sección global indica el nombre NetBIOS del servidor, la versión Samba que se está ejecutando, el grupo de trabajo o dominio al que pertenece el servidor y la lista de usuarios a los que se le permitirá el acceso.

La sección *printers* hace referencia a las impresoras del servidor Samba que compartiremos con los clientes. En el ejemplo, dado que no indicamos ninguna impresora concreta, vamos a compartir todas las impresoras conectadas al servidor. Además podrá imprimir cualquiera como invitado, aunque no tenga cuenta en el servidor.

Finalmente, en el fichero de ejemplo, existe una sección llamada *temp*, que identifica un recurso compartido llamado igual que el nombre de sección. En este caso, se comparte el directorio del servidor */home/temp*, con permisos de lectura y escritura, y los usuarios válidos serán *paco* y *virgilio*.

Si un cliente solicita un recurso no especificado en el archivo de configuración smb.conf, Samba intentará analizar si el nombre del recurso solicitado corresponde con un usuario que se intenta conectar o con una impresora del sistema, para ello deben aparecer las secciones *homes* y *printers* respectivamente.

8.6.4. Utilidades

Además de los demonios servidores comentados anteriormente, el paquete Samba nos ofrece una serie de aplicaciones cliente, de las que vamos a estudiar algunas en este apartado.

8.6.4.1. Programa smbmount

Permite montar un recurso compartido mediante SMB en un sistema Unix, algunas opciones del comando son:

- **username** : permite indicar el nombre del usuario con el que se accede al recurso.
- **password** : permite indicar el password del usuario con el que se accede al recurso.
- **fmask** : permite indicar los permisos que tomarán los ficheros (en formato numérico) en el directorio montado. De no especificarse se utiliza la máscara del shell actual.
- **dmask** : es similar a fmask pero para los directorios.
- **gid y uid** : permiten indicar los identificadores de grupo y de usuario respectivamente, asignados al recurso montado. Por defecto se asumen el ID del usuario y del grupo actuales (uid y gid).
- **rw y ro** : indican si el recurso se monta para lectura y escritura, o para lectura solamente.

Un ejemplo de utilización de la aplicación cliente podría ser el siguiente:

```
smbmount //estudiante/temp /mnt/temp -o username=juan, fmask=700, ro
```

8.6.4.2. Utilidad nmblookup

El comando nmblookup básicamente permite hacer consultas acerca de los nombres NetBIOS en una subred. Sin opciones, el comando traduce un nombre NetBIOS a dirección IP.

La sintaxis del comando se define de la siguiente manera:

```
nmblookup [opciones] <nombre_netbios>
```

La opciones aplicables al comando son:

- -A. Indica que se interprete el argumento como una dirección IP devolviendo el estado del nodo del *host* correspondiente, o sea todos los nombres *NetBios* registrados por dicho *host*.
- -T. Solicita que se realice una consulta inversa al DNS a partir del número IP obtenido.
- -M. Ordena que se busque el "*Master Browser*" del nombre NetBIOS dado como argumento.
- -S. Indica que además de devolverse la dirección IP se muestre el estado del nodo correspondiente, o sea todos los nombres NetBIOS registrados por el *host*

8.6.4.3. Aplicación smbclient

Permite acceder a recursos compartidos por un servidor SMB con una interfaz similar a la de los clientes FTP, estableciendo una conexión con dicho servidor. Una vez establecida la conexión SMB, se pueden trasladar, borrar e imprimir ficheros, entre otras funcionalidades.

Mediante el empleo de *smbclient* también es posible hacer consultas de lo compartido a través del protocolo por un servidor determinado, además de poder utilizarse para enviar mensajes mediante el protocolo *WinPopup*. La sintaxis de la orden es la siguiente:

```
smbclient [servicio] [password] [opciones]
```

Las distintas opciones de la sintaxis anterior tienen el siguiente significado:

- servicio. Se indica cuando se quiere establecer una conexión SMB con un determinado recurso compartido. Toma la forma //servidor/recurso. Donde servidor es el nombre NetBIOS del servidor y recurso es el nombre de lo compartido por dicho servidor. Para resolver el nombre del servidor se pueden emplear varios métodos en un determinado orden. Dichos métodos son mencionados más adelante. Para identificar al servidor también puede emplearse el número IP.
- Password. Es la contraseña que se utiliza para conectarse al recurso compartido. No existe una contraseña por defecto por lo que de no especificarse, el cliente siempre la solicitará aunque el recurso a acceder no la requiera. En este caso se presionará ENTER para indicar un *password* vacío. Con la opción -N, descrita más adelante, se puede evitar la pregunta.
- Opciones. Son las opciones que describen como se accede al recurso o que indican otra acción a realizar.

En este último caso, as opciones disponibles son:

- -U <usuario>. Permite especificar el usuario con el que se quiere establecer la conexión. En su defecto se utiliza el valor de la variable del entorno USER, y si esta no tiene valor se emplea el usuario GUEST. Solo es importante su utilización cuando se accede a recursos restringidos por usuario, por ejemplo para los servidores Windows NT.
- -L <host>. Permite consultar a un *host* servidor del protocolo acerca de todos los recursos que comparte. Además muestra los dominios o grupos de trabajo accesibles para este, así como el PDC5.1 de cada uno de los dominios. Para referirse al *host* se puede emplear su nombre *Netbios* o número IP. En este caso no se especifica ningún servicio.
- -N. Evita que se pregunte el *password* al establecer la conexión.
- -I. Permite especificar la dirección IP del servidor. Es útil cuando el mecanismo de resolución de nombres no devuelve la dirección deseada.

- -M <host>. Permite enviar mensajes a través del protocolo *WinPopup* a un <host> que tenga activado el servicio. El contenido del mensaje se toma de la entrada estándar durante la ejecución del comando.

8.7. CASO DE USO 3: CODA

Sistema de archivos en red avanzado, basado en el sistema de archivos AFS2 desarrollado en la prestigiosa universidad Carnegie Mellon. Un sistema de archivos distribuido almacena los datos en uno o varios servidores de ficheros, y los hace accesibles a los clientes de red. Las ventajas de un enfoque distribuido son, entre otras, la disponibilidad de los datos desde distintas computadoras a las cuales se les puede distribuir una copia desde los servidores, la capacidad de almacenamiento de los servidores que habitualmente es muy superior a la de los clientes, la realización de copias de seguridad sólo de los servidores de archivos y, sobre todo, la facilidad para que un grupo de usuarios compartan y utilicen documentos.

Pero el uso de sistemas distribuidos tiene también una serie de inconvenientes. La transferencia de archivos por la red implica penalización en su rendimiento y consumo de ancho de banda. La seguridad es un aspecto muy importante, debemos estar seguros de que los clientes están realmente autorizados para acceder a la información. Finalmente, existen problemas relacionados con la conectividad, clientes que pierden la conexión con el servidor voluntaria o involuntariamente, o fallos en el servidor que dejen a todos los clientes desconectados.

Coda es un sistema de archivos distribuido que intenta abordar o solucionar los inconvenientes citados en el párrafo anterior, intenta reducir el tráfico de red, aportar mecanismos de seguridad y, principalmente, aborda aspectos relacionados con la desconexión de la red. Este sistema de archivos posee diversas características avanzadas.

- Altas prestaciones mediante caché persistente en el cliente que se sincroniza al realizar escrituras. Siempre que sean lecturas se estará trabajando en la cache local, sin necesidad de conectar con el servidor. La cache en el cliente aumenta el rendimiento en los clientes y reduce el ancho de banda necesario por el menor tráfico de datos por la red.
- Replicación de servidores. Esta característica le aporta una gran tolerancia a los fallos, ya que en el caso de caer un servidor otro continuará el servicio automáticamente.

- Buena escalabilidad gracias a la replicación y a la partición de servidores.
- Adaptación al ancho de banda
- Funcionamiento continuo ante perdidas de conexión
- Funcionamiento sin conexión (equipos móviles). Permite continuar funcionando aunque se esté desconectado, realizando una reintegración de los datos desde los clientes cuando se vuelva a conectar.
- Modelo de seguridad para autenticación, encriptación y control de acceso. Puede utilizar mecanismos de autenticación como Kerberos y listas de control de accesos.

8.7.1. Conceptos

En el apartado siguiente veremos la arquitectura de coda y su funcionamiento, vamos a analizar en este apartado una serie de conceptos que nos ayudarán a entender el funcionamiento de Coda.

8.7.1.1. Celda Coda

El concepto de *celda* en Coda se utiliza para definir un conjunto de servidores que comparten una base de datos de configuración común. De entre todos los servidores, uno tiene que ser la máquina que controla el sistema (SCM – System Control Machine) que se encarga de las siguientes funciones:

- Equipo que controla el sistema, entendiendo por sistema a la celda a la que pertenece el SCM.
- Único equipo que modifica la configuración
- Propaga la configuración al resto

Actualmente, un cliente coda solo puede trabajar con una celda, aunque se espera que en futuras implementaciones se pueda conectar a varias celdas simultáneamente.

8.7.1.2. Volúmenes

En muchos sistemas de ficheros de red el servidor disfruta de una estructura estándar de archivos y exporta un directorio a los clientes, tal y como hemos analizado en el sistema de archivos de red o NFS. En Coda la organización es algo distinta.

Los archivos en los servidores Coda no se almacenan con la estructura de un sistema de ficheros tradicional. Las particiones en los ordenadores servidores Coda pueden ponerse como disponibles al servidor de ficheros. Esas particiones contendrán archivos que son agrupados en volúmenes. Cada volumen tiene una estructura de directorios como en un sistema de ficheros estándar: un directorio raíz del volumen y un árbol bajo el directorio raíz.

Un volumen completo es mucho más pequeño que una partición, pero mucho mayor que un simple directorio y es una unidad lógica de archivos. Por ejemplo, el directorio *home* de un usuario podría ser normalmente un único volumen Coda.

8.7.1.3. Puntos de montaje

De manera similar al montaje que se hace en NFS, los clientes de Coda montan volúmenes. Existe un volumen principal que se monta en /coda, y que es montado en el arranque del sistema o del proceso de administra la cache del cliente, denominado *venus*.

A partir del montaje inicial, podemos ampliar el árbol insertando otros volúmenes. Para montar nuevos volúmenes a partir del principal /coda se utiliza el comando *cfs mkmount*, que recibe los siguientes parámetros:

- nv. nombre del volumen
- pa. Punto de anclaje

Existen dos diferencias de los puntos de anclaje en Coda con respecto a Linux o Unix. Por una parte, el punto de anclaje tiene que ser un directorio que no exista, en el caso de Linux es todo lo contrario. Por otra parte, el anclaje es persistente, es decir, no es necesario volverlo a montar después de reiniciar la máquina cliente.

8.7.2. Arquitectura

La arquitectura del sistema se puede observar en la figura 8.9. Como cualquier otro sistema de ficheros, un ordenador habilitado para usar el sistema de ficheros de Coda necesita algún tipo de soporte en el núcleo para acceder a los archivos de Coda. El soporte para Coda del núcleo es mínimo, y trabaja en unión con el administrador de la caché en el espacio de memoria de usuario *venus*. Los programas usuarios realizarán peticiones al núcleo, que responderá directamente o preguntará al administrador de la caché para que le ayude a ofrecer el servicio.

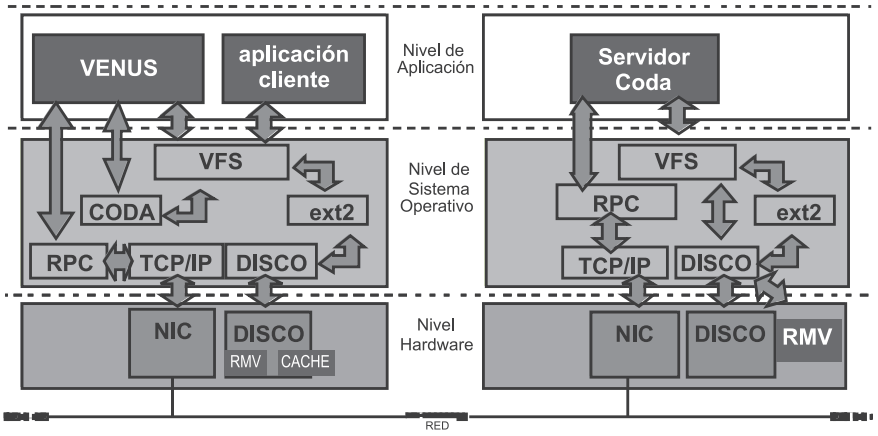


Figura 8.9. Arquitectura del sistema CODA

Uno de los aspectos más interesantes que podemos observar en la arquitectura Coda es que, al igual que el sistema de archivos de red NFS, en la capa del sistema operativo funciona sobre el protocolo de llamada a procedimiento remoto RPC. El cliente *venus* realizará la petición al servidor a través del protocolo RPC, mientras que el servidor de Coda responderá igualmente mediante RPC.

Como hemos comentado en el apartado anterior, todos los archivos de cualquiera de los servidores de la misma celda están disponibles en el cliente bajo el punto de montaje /coda, por lo que todos los clientes ven el mismo espacio de nombres. Ésta característica tiene la ventaja de que todos los clientes se pueden configurar de la misma forma, y los usuarios siempre verán el mismo árbol de directorios. Con NFS los clientes necesitan una lista actualizada de los servidores y de los directorios que exportan.

Cuando una aplicación cliente (figura 8.9) solicita acceder a un archivo Coda por primera vez, la llamada pasará al núcleo y será atendida por el sistema virtual de archivos VFS. VFS detectará que la llamada corresponde a un archivo Coda, pasará la llamada al módulo del sistema de archivos Coda en el núcleo, y este a su vez reenviará la petición al administrador de cache *venus*. El administrador de cache buscará el archivo solicitado en la cache de disco, y en caso de no encontrarlo contactará con los servidores de su celda para solicitar el archivo.

El administrador de cache *venus* obtendrá el archivo de los servidores utilizando el protocolo de comunicación RPC. Una vez obtenido el archivo lo almacenará en la caché persistente local, y a partir de ese momento se accederá a él como a cualquier otro archivo local, sin necesidad de contactar con los servidores. Si se abre el archivo otra vez, no será necesario solicitarlo a los servidores, por estar localmente en la memoria caché. La caché local es un contenedor de archivos almacenados en */usr/coda/venus.cache*.

Los archivos de directorio y los atributos de los archivos también los obtiene *venus* y los almacena de forma local sin formatear en un archivo especial de memoria virtual recuperable (RVM – Recoverable Virtual Memory) llamado */usr/coda/DATA*. Si un archivo se modifica y se cierra, *venus* actualiza los servidores enviándoles el nuevo archivo. Otras operaciones como la creación de directorios, el borrado de archivos y la creación o borrado de enlaces, también son propagadas a los servidores. En otras palabras, Coda sitúa en la cache toda la información que necesita el cliente y sólo informa la servidor de los cambios que deben realizarse en el sistema de archivos.

Por otra parte, en el lado del servidor se necesita más información o metadatos para gestionar los archivos compartidos: propietarios, control de acceso, versiones o atributos de los archivos. Esta información se guarda en particiones sin formatear accesibles a través del paquete de memoria virtual recuperable RVM, mientras que los datos de los archivos residen en particiones formateadas del servidor.

Si se modifica un archivo se propaga al servidor de forma síncrona, pero, si se detecta que no hay conexión se pasa a modo *desconectado*. En este modo los cambios se almacenan en el registro de modificaciones del cliente o CML y al reconectar se usa el CML para actualizar el servidor.

Cuando se reintegra el CML de los clientes en los servidores pueden aparecer conflictos de consistencias que deben resolverse, por ejemplo, otro cliente puede haber modificado el mismo archivo que nosotros. Se necesita un sistema de resolución de conflictos que, en el caso de Coda, puede ser automático o manual. Automático si, por ejemplo, se han modificado registros distintos de una base de datos. Si no se puede resolver de manera automática, será necesaria la intervención del administrador para resolver el conflicto manualmente.

8.7.3. Servicios

El funcionamiento de este sistema de archivos de alto rendimiento se basa en la ejecución de los siguientes procesos servidores o subsistemas de Coda (figura 8.10):

- codasrv. Es el servidor principal de Coda, atiende las peticiones de archivos de los clientes y lleva a cabo todas las tareas de gestión del sistema distribuido apoyándose en el resto de servicios. Realizará la resolución de conflictos cuando se puedan producir inconsistencias en los archivos. Solicitará a *auth2* la autenticación de los usuarios cuando acceden al sistema, los usuarios autenticados recibirán un testigo que les permitirá solicitar servicios a los distintos servidores sin necesidad de autenticarse en cada uno de ellos. A través de *volutil* llevará a cabo la creación, modificación y eliminación de volúmenes. Mediante el demonio *backup* realizará la replicación de los servidores, lo cual aporta alta disponibilidad de los datos, si un servidor falla, otro tomará el control y realizará el servicio al cliente. Finalmente, después de modificar la base de datos de configuración, actualizará la base de datos en todos los servidores de la celda mediante el servicio *update*.
- auth2. Se encarga de la autenticación de los clientes y la gestión de testigos.
- volutil. Servicio utilizado por codaserv para la gestión de los volúmenes.

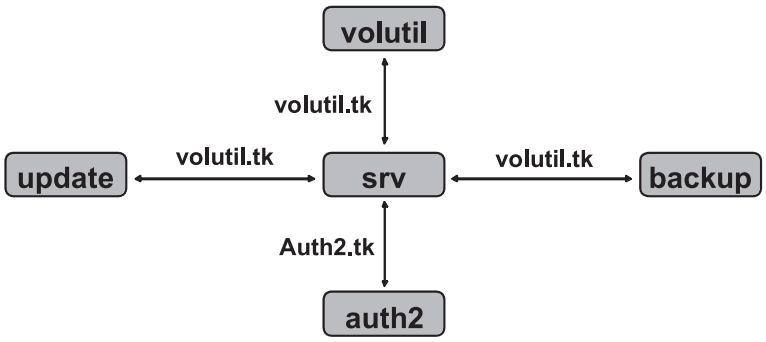


Figura 8.10. Subsistemas de los servidores Coda.

- backup. Demonio que permite la realización de copias de seguridad del sistema y, quizás más importante, la replicación de los servidores.
- update. Servicio utilizado por el SCM para comunicar al resto de servidores a su cargo la actualización de la base de datos de configuración.

8.7.4. Instalación

Coda está disponible tanto en código fuente como en paquetes rpm, y existe soporte en el núcleo para este sistema de archivos a partir de la versión del núcleo 2. Para la instalación de los paquetes se deben seguir distintos pasos, dependiendo de si estamos en los clientes o en los servidores.

Para la instalación de los servidores tenemos que instalar los siguientes paquetes RPM:

```
coda-debug-server-XX.i386.rpm
coda-doc-XX.i386.rpm
```

Después de instalar los paquetes anteriores, tenemos que configurar la instalación mediante el comando siguiente:

```
vice-setup
```

En este punto, la instalación habrá terminado y podremos ejecutar los demonios de Coda en los servidores mediante el script de arranque *vice.init*. Debemos tener en cuenta que para configurar un servidor SCM necesitamos un directorio vacío */vice* donde el servidor de archivos pondrá los ficheros y una partición sin formatear para los metadatos RVM.

En cuanto a la instalación de los clientes, es necesario instalar los siguientes paquetes:

```
coda-debug-client-XX.i386.rpm
coda-fs-module-kXX.i386.rpm
coda-doc-XX.i386.rpm
```

Después de instalar los paquetes anteriores, tenemos que configurar la instalación en el cliente mediante el comando siguiente:

```
venus-setup
```

Ahora deberíamos tener instalado Coda en el cliente, el módulo de soporte del núcleo y la aplicación a nivel de usuario para la administración de la caché *venus*. Podemos comprobar que el núcleo de sistema operativo soporta Coda en el archivo */proc/filesystems*.

8.7.5. Administración

Para la administración completa de Coda se utilizan más de treinta archivos de configuración, algunos de los cuales se acceden a través de guiones o scripts. Como hemos estudiado en apartados anteriores, Coda está formado por distintos subsistemas, éstos subsistemas comparten todos los datos de configuración, datos que son distribuidos por todos los servidores dependientes de la misma celda Coda, o lo que es lo mismo, dependientes del mismo SCM.

No vamos a ver todos los archivos involucrados en la configuración completa de Coda, analizaremos el principal archivo de configuración. Sin embargo, es importante resaltar que mediante la instalación correcta del sistema de archivos distribuido, tanto vice-server-setup como venus-setup construyen una configuración del sistema básica. Los distintos archivos de configuración de Coda están bajo el directorio */vice/db*.

El principal archivo de configuración es el conocido como base de datos de Coda, tiene el formato del listado 8.6. En el archivo, al igual que en otros analizados anteriormente, se pueden identificar distintas secciones. La primera y más importante sección es la identificada como *global*, en ésta sección se identificarán las directivas o parámetros que afectan a la celda completa. En el ejemplo podemos observar la directiva *rootvolume* encargada de identificar la raíz del volumen, ésta directiva la podemos indicar directamente en el archivo de la base de datos o se la podemos indicar en el proceso de instalación del servidor descrito en el apartado anterior. La etiqueta *cell* nos permitirá indicar el nombre de la celda a la que se podrán conectar los clientes, recordar que hasta el momento Coda sólo permite a un cliente conectarse a una única celda. Con *scm* podremos indicar el nombre de la máquina que se encargará de controlar el sistema, en el ejemplo se encargará una máquina llamada *lactea*. Finalmente, dentro de la sección global del ejemplo, podemos observar la directiva *servers*, identificando una lista de servidores de Coda que pertenecen a la misma celda.

Luego podemos observar secciones asociadas a cada máquina perteneciente a la celda, en el ejemplo se muestran opciones para la máquina *lactea*. Dentro de la sección asociada a la máquina se crean otras secciones

```
[global]
    ROOTVOLUME = coda.root
    cell = dtic.es
    scm = lactea
    servers = lactea, sanpedro, ….

[lactea]
    [partitions]
       /vicepa raw_type
       /vicepb tree_type depth=5,width=4
    [rvm]
       data=/dev/sda5
       datasize=10000000

[sanpedro]
       . . .
```

Listado 8.6. Base de datos de Coda.

que nos permiten diferenciar entre las particiones asociadas a los ficheros de datos y aquellas particiones que reservamos para los metadatos accesibles a través de RMV.

En el ejemplo podemos observar dos particiones de datos, una sin formato específico y la otra formateada con unas dimensiones máximas. En cuanto a la partición dedicada a los metadatos será */dev/sda5* y tendrá un tamaño de 10 megas. Para conocer las distintas directivas y opciones de configuración debemos consultar la documentación de Coda.

8.7.6. Utilidades

El sistema de archivos distribuidos CODA proporciona, además de los servicios, las utilidades que se describen a continuación.

- cfs. Constituye la principal utilidad disponible en el cliente para interactuar con los servidores. Una de sus principales utilidades, como se ha estudiado en un apartado anterior, es montar nuevos volúmenes debajo del volumen principal del cliente */coda*. Permite obtener información de la caché persistente local y solicitar las listas de control de acceso.

- clog. Permite al cliente autenticarse en el servidor de autenticación de Coda *auth2* y obtener el testigo que le permitirá acceder a los archivos.
- Codacon. Utilidad usada en el cliente para monitorizar las operaciones de administración de la caché persistente.
- cmon. Utilidad del cliente para conseguir la lista de servidores pertenecientes al mismo cluster o celda.

9. SERVICIOS DE DIRECTORIO

Un *servicio de directorio* se define como el conjunto de aplicaciones, protocolos y sistemas de almacenamiento que permite archivar, gestionar y dar acceso a un conjunto de procesos clientes a una información que es organizada de forma jerárquica. Un servicio de directorios tiene muchas similitudes con un servicio de base de datos, si bien el servicio de directorio tiene algunas características que lo diferencian de éste.

Por *directorio* entendemos la base de datos que da soporte a los objetos que son gestionados mediante el servicio de directorio. En un directorio la información almacenada es descriptiva y está basada en atributos que

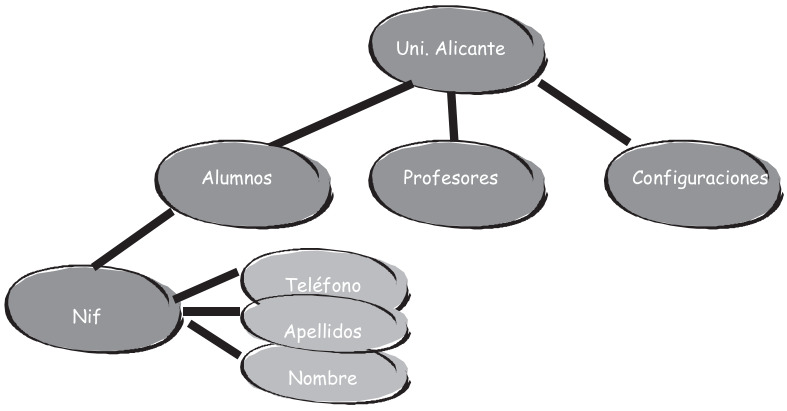


Figura 9.1. Ejemplo de directorio

son asociados objetos que se gestionan (ver figura 9.1). Estos objetos se estructuran jerárquicamente dentro del directorio en función del uso que se quiera dar a éste. Debido a esta característica los directorios suelen utilizarse para almacenar información referente a organizaciones, usuarios, grupos, redes de computadores, recursos, etc., donde la relación jerárquica entre estos elementos es fundamental para su gestión.

El servicio de directorio actúa como interfaz entre el directorio y los nodos que quieren acceder a éste, siendo el responsable de validar el acceso y efectuar las operaciones solicitadas por los usuarios (consultas, inserciones, borrados, modificaciones, etc.).

9.1. CARACTERÍSTICAS DE LOS DIRECTORIOS

Una de las principales características de los directorios, y base fundamental para su diseño, es que están optimizado para las operaciones de lectura y búsqueda de información, siendo las operaciones de inserción, borrado y actualización mucho menos eficientes.

La información almacenada en el directorio está definida mediante un esquema que, al contrario que en el caso de las bases de datos, es fácilmente extensible y modificable. Esto permite modificar o incorporar atributos a los objetos del directorio sin que esto implique reestructuraciones importantes en su estructura.

El servicio de directorios utiliza un modelo distribuido que permite almacenar la información de forma distribuida donde diversas partes del directorio (subárboles) son gestionadas por diferentes servicios de directorio. El modelo también facilita la replicación de la información (servidores secundarios) de forma que podemos tener todo o parte del directorio replicado para ganar en eficiencia y escalabilidad. Para alcanzar los altos niveles de eficiencia (principalmente para las lecturas), este esquema normalmente se implementa con características de débil consistencia, permitiéndose inconsistencias temporales mientras se propaga, por ejemplo, una actualización o inserción de registro.

9.2. DIRECTORIOS Y BASES DE DATOS RELACIONALES

Si bien como hemos comentado existen similitudes entre los servicios de directorio y de base de datos existen varias características que los distinguen.

En un servicios de directorio se realizan muchas más operaciones de lectura que de escritura, que son prácticamente residuales, por lo que no es necesario implementar operaciones de *rollback*. Por esta misma razón en un directorio no son necesarias las transacciones ni las operaciones de bloqueo de registro.

En un directorio, por su estructura jerárquica, es posible que, al contrario de lo que suele ocurrir en una base de datos relacional, la información esté duplicada en algunas secciones, si bien esto se considera un mal menor en pro de la eficiencia en las lecturas.

En cuanto a la estructuración interna de los datos se pueden establecer algunas analogías entre directorios y bases de datos. En los directorios los objetos serian similares a los registros de una tabla en una base de datos relacional, siendo los atributos del objeto similares a las columnas de una tabla. En un directorio cada objeto viene definido por un *tipo de objeto* definido que sería lo más parecido a la definición de un tabla en una base de datos. Para cada tipo de objeto se establecen un conjunto de atributos (similares a las columnas de una tabla). En la definición de un tipo de objeto algunos atributos son considerados obligatorios, es decir que todos los objetos de ese tipo tienen que tener estos atributos, y un conjunto de atributos optativos, que cada objeto pude definir o no. Otra de las grandes diferencias es que en los directorios los atributos son multivaluados, es decir, que pueden tener más de un valor. Por ejemplo si tenemos un objeto *persona* con un atributo *teléfono* este podría tener dos valores, uno por cada teléfono de dicha persona. Otra característica a destacar en los directorios es que cada objeto del mismo puede estar asociado a más de un tipo de objeto, aplicándose en ese caso los atributos de todos los tipos a los que pertenece.

Los servicios de directorio están pensados para ser accedidos desde un número elevado de clientes y altamente heterogéneos. Los datos se pueden distribuir de mejor forma que en una base de datos donde, por ejemplo, normalmente no se puede o es muy complejo dividir la gestión una misma tabla entre distintos servidores.

9.3. EJEMPLOS DE SERVICIOS DE DIRECTORIOS

Domain Name System (DNS), comúnmente utilizado para relacionar nombres de dominios a direcciones IP, es posiblemente el servicio de directorio más extendido en internet. Se trata de una base de datos distribuida y jerárquica que gestiona información relacionada con nombres de dominio.

Network Information Service (NIS) nos aporta un servicio de configuración de red, originalmente pensado para redes Unix, que permite centralizar información referente a equipos y usuarios de la red.

X.500 fue uno de los primeros servicios de directorio propuesto de carácter general y fue desarrollado conjuntamente por ISO y el CCITT. Su protocolo especifica un modelo de conexión de servicios de directorio locales para formar un directorio global distribuido en Internet

En la actualidad LDAP, que surgió como superación de X.500, es el servicio de directorio de carácter general más extendido. Sus dos implementaciones más extendidas, *OpenLDAP* en los entornos Unix y *ActiveDirectory* en entornos Windows, son en la actualidad las principales herramientas para la configuración y gestión de redes de computadores. Existen otras implementaciones de LDAP como son *Novell Directory Services* (*eDirectory*), *iPlanet Directory Server* y *Red Hat Directory Server*.

9.4. PROTOCOLO DE ACCESO A DIRECTORIO

LDAP (Lightweight Directory Access Protocol) en un protocolo de aplicación sobre TCP/IP que permite el acceso a un servicio de directorio. Los directorios con los que trabaja LDAP son de propósito general si bien es utilizado comúnmente para almacenar información referente a organizaciones, elementos en redes de computadores, usuarios o documentos.

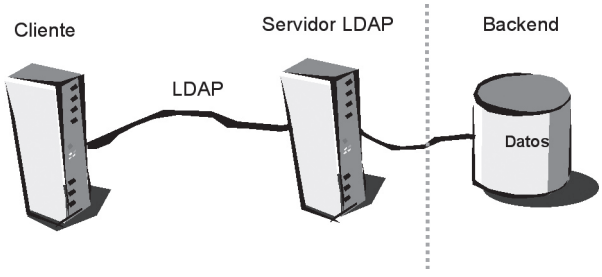


Figura 9.2. Esquema de LDAP

La actual versión de LDAP, la versión 3, está definida, al igual que otros protocolos TCP/IP, en una serie de documentos de especificación del IETF (*Internet Engineering Task Force*) como se describe en la tabla 9.1.

RFC	Nombre
2251	Lightweight Directory Access Protocol (v3)
2252	Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions
2253	Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names
2254	The String Representation of LDAP Search Filters
2255	The LDAP URL Format
2256	A Summary of the X.500(96) User Schema for use with LDAPv3
2829	Authentication Methods for LDAP
2830	Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security
3377	Lightweight Directory Access Protocol (v3): Technical Specification

Tabla 9.1. RFC's de especificación de LDAPv3

LDAP nace, y de ahí su nombre, como una alternativa "ligera" al protocolo X.500 que implementa un protocolo basado en la pila OSI, conocido como DAP, que es mucho más pesada que la pila TCP/IP de LDAP (ver figura 9.3). Si es necesario, un cliente LDAP puede acceder a un servicio de directorio DAP mediante una pasarela LDAP-to-DAP.

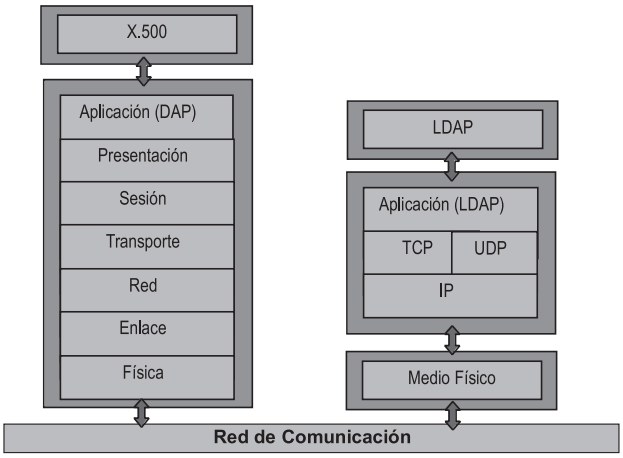


Figura 9.3. Pila de protocolos de X.500 y LDAP

Si pensamos en una aplicación web con sus páginas HTML y su información gráfica asociada, almacenados en un servidor Web, y cuya información suele ser de sólo lectura podríamos pensar que esta información es idónea para ser almacenada en un servicio de directorio. De hecho se considera que ya se utiliza un servicio de directorio especial para estos recursos el sistema de archivos, en este caso del servidor web. LDAP no supone un reemplazo para directorios especializados como el caso de los sistemas de archivo o DNS.

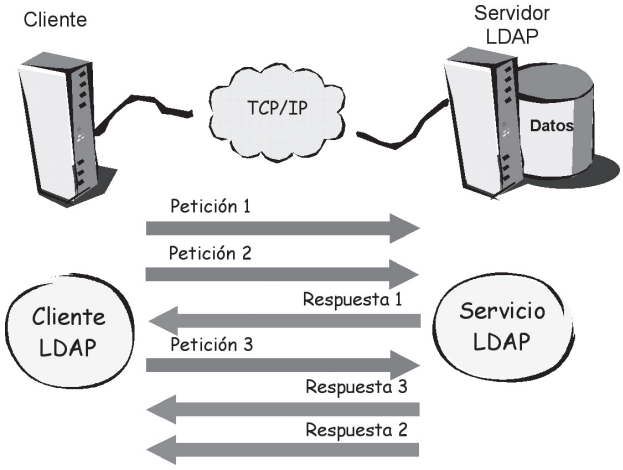


Figura 9.4. Esquema del paso de mensajes asíncronos en LDAP

Otro ejemplo sería un servidor de impresión. LDAP no es válido para almacenar los archivos listos para su impresión pero sí podría ser válido para almacenar la información de configuración de impresoras en red.

Aunque puede ser útil almacenar determinada información binaria en LDAP, como carácter general, no es adecuado para almacenar este tipo de información pero sí que es muy adecuado para almacenar configuraciones.

9.4.1. Definición del protocolo

LDAP es un protocolo de aplicación sobre TCP/IP y que se articula mediante un esquema cliente/servidor basado en paso de mensajes. El uso del directorio comienza con una conexión del cliente, normalmente en

el puerto TCP por defecto 389. El cliente realiza solicitudes al servidor mediante el uso de un limitado conjunto de operaciones siendo este proceso asíncrono, es decir el cliente puede enviar varias peticiones antes de obtener las respuestas del servidor, pudiendo llegar estas de forma desordenadas. La figura 9.4 ilustra el funcionamiento del paso de mensajes de LDAP.

9.4.2. Modelos LDAP

En LDAP se pueden distinguir cuatro modelos que representan los servicios que proporciona un servidor LDAP vistos por el cliente.

El *modelo de información* establece la estructura y los tipos de datos que tiene el directorio: esquemas, entradas, atributos, etc. Según este modelo un directorio está formado por entradas estructuradas en forma de árbol (cada entrada tiene asociada la entrada de su nodo padre). Cada entrada estará definida por un conjunto de atributos. Cada atributo es un par nombre y valor. El conjunto de atributos se define en el esquema.

El *modelo de asignación de nombres* define cómo referenciar de forma única las entradas y los datos en el árbol de directorios. Cada entrada tendrá un identificador único llamado DN (*Distinguished Name*). El DN se construye a partir de un RDN (*Relative Distinguished Name*) que se compone de varios atributos de la entrada, seguido de los DN de sus ancestros.

El *modelo funcional* establece las operaciones para acceder al árbol de directorio: autenticación, solicitudes y actualizaciones.

Operación	Descripción
Bind	Auténtica el cliente y especifica la versión del protocolo
Start TLS	Proteje la comunicación mediante Transport Layer Security (TLS)
Search	Realiza una búsqueda en el directorio
Compare	Verifica si una entrada tiene un valor de atributo
Add	Añade una nueva entrada
Delete	Borra una entrada
Modify	Modifica una entrada
Modify DN	Mueve o renombra una entrada
Abandon	Aborta solicitudes previas
Extended Operation	Operación genérica utilizada para definir otras operaciones
Unbind	Cierra la conexión

Tabla 9.2. Operaciones comunes en LDAP

Por último el *modelo de seguridad* establece los mecanismos que garantiza para el cliente cómo probar su identidad (autenticación) y para el servidor cómo controlar el acceso (autorización).

Tipo	Descripción
bin	Binario
ces	cadena con mayúsculas y minúsculas exactas (las mayúsculas y minúsculas son significativas durante las comparaciones)
cis	cadena con mayúsculas y minúsculas ignoradas (las mayúsculas y minúsculas no son significativas durante las comparaciones)
tel	cadena de número de teléfono (como cis, pero durante las comparaciones se ignoran los espacios en blanco y los guiones "-")
dn	nombre distintivo (distinguished name)

Tabla 9.3. Tipos de datos para los atributos LDAP

9.4.3. Elementos del modelo de información

Las entradas son las unidades básicas de LDAP equivalentes a los nodos que conforman un árbol. Cada una de ellas refleja un concepto u objeto del mundo real (usuarios, organizaciones, hosts,...). Cada entrada está definida jerárquicamente de forma relativa a su nodo padre.

Cada entrada está compuesta por una serie de atributos. Cada uno de estos se forma por pares `nombre:valor`. Cada atributo posee unas reglas de sintaxis en función de un *tipo*. En la tabla 9.5 se pueden ver los diferentes tipos existentes. El concepto de atributo es similar a las variables de programación. Los atributos pueden ser *multivaluados*, es decir, cada atributo puede tener más de un valor asignado.

En LDAP una entrada se puede definir mediante un archivo LDIF (*LDAP interchange format*) en el que se describen los diferentes atributos que lo componen. En cada archivo LDIF se pueden definir una o más entradas, usando una línea en blanco como separador entre entradas. En la figura 9.6 se puede ver un ejemplo de fichero LDIF. Los comentarios se establecen en el fichero mediante el símbolo almohadilla (`#`).

Se pueden distinguir dos tipos de atributos: *atributos atómicos* (`cn`, `dc`, `ou`, `sn`) y atributos de clase que indican el tipo de entrada que se está definiendo (`ObjectClass`).

Las clases definen los atributos que tiene un tipo de entrada determinado. Toda entrada tiene que tener al menos un atributo `objectclass` que indica la clase a la que pertenece, y por lo tanto el resto de atributos que debe y puede tener. Cada entrada puede tener más de un atributo `objectclass`.

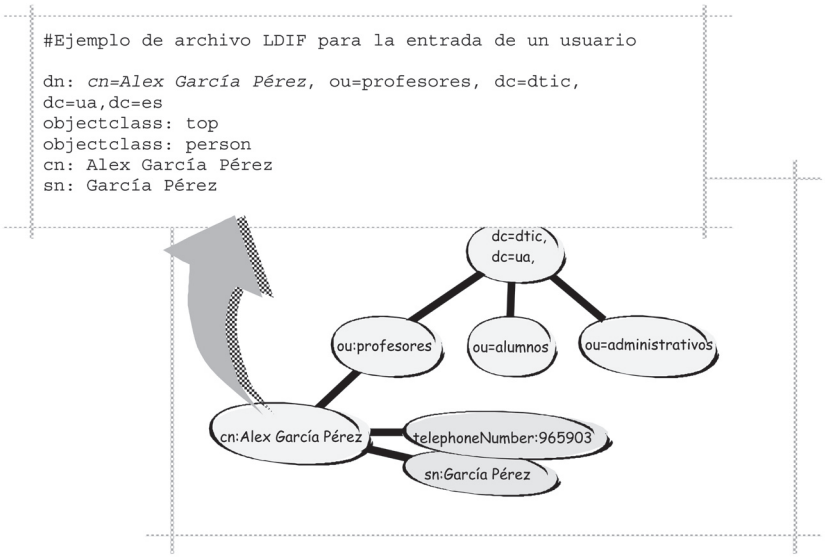


Figura 9.5. Ejemplo de fichero LDIF

En una clase se definen los atributos como requeridos (atributos obligatorios) o permitidos (atributos opcionales). Cada `ObjectClass` tiene un identificador único (OID - Object identifier).

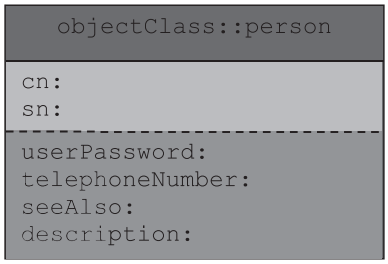


Figura 9.6. Definición de una Clase

Un esquema establece una platilla donde se definen todas las clases (`ObjectClass`) de un directorio, donde se definen los atributos y configuraciones prototípicas.

Cada entrada del directorio será identificada por un atributo especial de tipo **dn** que actuará como identificador único (*DN – Distinguished Name)* de esta. El DN estará conformado por los diferentes identificadores relativos (RDN – Relative Distinguished Name) del propio nodo y de sus antecesores. En el ejemplo de la figura 9.7 tenemos un nodo cuyo DN es:

```
dn: cn=Alex García Pérez, ou=profesores, dc=dtic, dc=ua,dc=es
```

y un RDN que es:

```
cn: Alex García Pérez
```

EL RDN siempre es el primer elemento del DN. En todo el directorio no puedes existir dos entradas con el mismo DN lo cual implica que dos entradas con el mismo *padre* no pueden tener el mismo RDN. Esto es similar al concepto de archivo y rutas relativas y absolutas en un sistema de archivos.

Los servidores LDAP utilizan sistemas de bases de datos como *back-storage* donde almacenar y gestionar las entradas del directorio.

9.4.4. Características distribuidas

Se puede dividir el árbol de directorios en subárboles de tal manera que diversos servidores LDAP controlen un subárbol por los siguientes motivos:

- Rendimiento: al distribuir el directorio en varios servidores distribuimos la carga individual de cada uno de ellos y por lo tanto ganamos en rendimiento global.
- Localización Geográfica: cada servidor puede dar servicio a una zona geográfica diferente.
- Cuestiones Administrativas: cada servidor es gestionado por administradores diferentes.

Para llevar a cabo dicha distribución cada subárbol o rama será referenciada desde el árbol padre mediante un registro especial de unión de

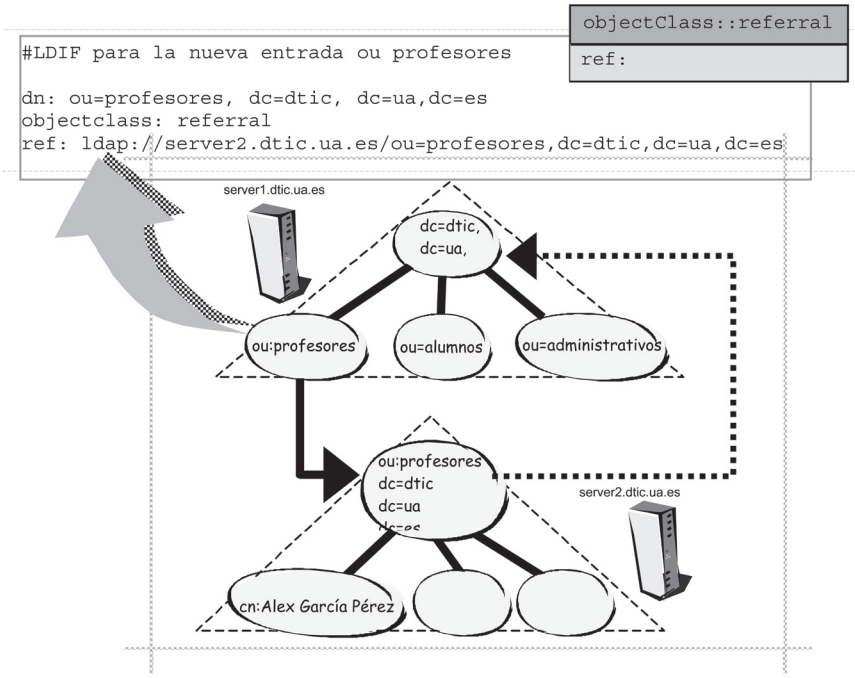


Figura 9.7. Ejemplo LDIF para realizar un enlace a otro servidor

tipo `objectClass::referral`. En la figura 9.8 se puede ver un ejemplo de esto, donde el servidor `server2.dtic.ua.es` gestiona el subárbol *profesores* del servidor que se enlaza desde `server1.dtic.ua.es`.

El modelo distribuido tiene dos modos de funcionamiento: el servidor LDAP padre (al que solicita un valor el cliente) resuelve la solicitud comunicándose con el LDAP hijo y devolviendo el valor al cliente, o bien el servidor padre indica al cliente que servidor tiene el dato y es el cliente el que vuelve a conectarse con el servidor hijo para solicitar nuevamente el dato.

9.4.5. Seguridad

Dado que LDAP es un protocolo orientado a la conexión y basado en intercambio de mensajes existe un peligro potencial en el trasiego de información y se hace necesario tanto controlar la conexión de los clientes como el cifrar el trasiego información.

Para ello todas las operaciones se controlan mediante el nivel de autorización establecido para el usuario autenticado. La autenticación mediante un identificador de usuario (*login*) y una contraseña (previamente cifrada con CRYPT, MD5, SHA, SSHA, etc.). Estos valores se almacenas en registros especiales de LDAP de tipo `objectClass::person` (ver).

Los diferentes mecanismos de autenticación que existen son:

- Anónima: No se realiza autenticación y da acceso a información pública.
- Simple: Especifica mediante un DN un usuario y aporta la contraseña necesaria para ese usuario. Es similar a la validación simple de HTTP. En este caso la contraseña viaja sin codificar.
- Simple sobre SSL/TLS: Previamente se negocia una capa de transporte seguro y la información viaja de forma segura.
- Simple y Capa de Seguridad (SASL): Utiliza un módulo configurable que permite negociar previamente el mecanismo de autenticación y la capa de transporte seguro (kerberos, GSSAPI, S/Key, EXTERNAL).

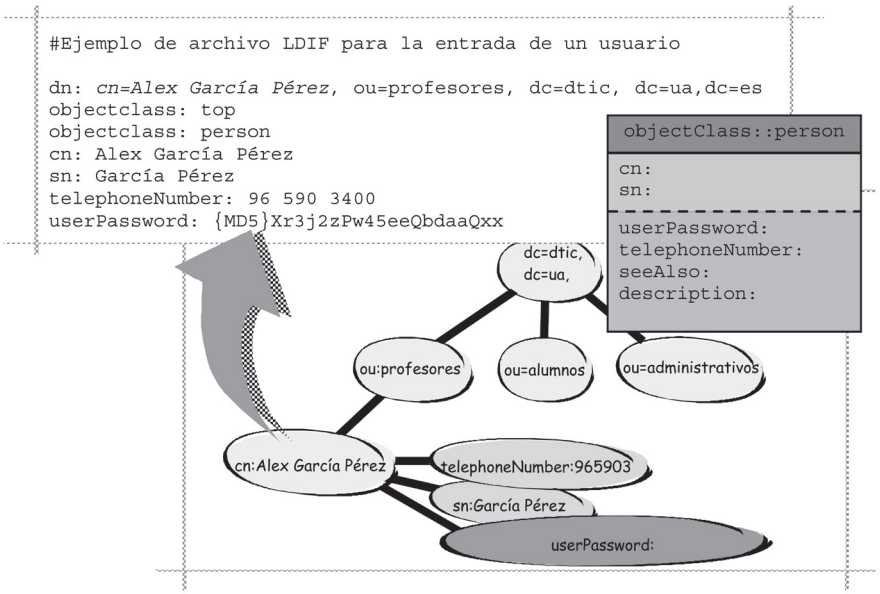


Figura 9.8. Ejemplo de archivo LDIF para la entrada de un usuario

9.5. CASO DE USO: OPENLDAP

OpenLDAP es una implementación de código abierto de LDAP desarrollada por OpenLDAP Project. Las características más destacadas de OpenLDAP son:

- Se distribuye bajo licencia libre pudiendo tener acceso al código fuente.
- Es multiplataforma existiendo versiones para Linux, Windows, BSD, HP-UX, Solaris, Mac OS X y z/OS.
- Tiene una buena integración con multitud de aplicaciones, principalmente en el mundo Linux.
- Soporta la especificación LDAPv3.
- Soporta IPv6.
- Soporte de Referrals (esquema distribuído).
- Soporte para la replicación.
- Permite operaciones de publicación de esquemas antes de realizar búsquedas.
- Internacionalización mediante caracteres UTF-8 y atributos de lenguaje.
- Soporta multitud de bases de datos como almacén de datos.
- Permite mecanismos SASL y SSL/TLS.
- Soporta extensiones en el protocolo (modificación y creación de nuevas operaciones).
- Contiene un esquema de mapeo entre Radius y OpenLDAP.
- Tiene mecanismos avanzados de búsqueda.

OpenLDAP puede ser obtenido de la página web oficial www.openldap.org.

Los principales componentes de OpenLDAP son:

- slapd: El servidor (demonio) principal de LDAP
- slurpd: El servidor de replicación de LDAP. Permite sincronizar varias replicas de un servidor LDAP.
- El conjunto de librerías que implementa el protocolo LDAP.
- Un conjunto de herramientas, utilidades y aplicaciones cliente.

9.5.1. Prerrequisitos de la instalación

Antes de realizar la instalación del servidor OpenLDAP, y en función de las necesidades de uso del servicio, es posible que previamente se necesite instalar ciertos paquetes opcionales de terceras partes.

9.5.1.1. Software de base de datos

OpenLDAP utiliza por defecto una base de datos *Sleepycat Software Berkeley DB* como base de datos primaria (*back-storage*). Si este software no esta instalado en el equipo previa instalación de OpenLDAP ésta no podrá llevarse a cabo, dando un error en el proceso de configuración.

Se puede obtener una distribución de la base de datos Berkeley en la dirección `http://www.sleepycat.com/download/`. Una vez descargada la última versión estable desempaquetar usando la siguiente sentencia reemplazando VERSION por la versión del archivo descargado:

```
tar xfvz db-VERSION.tar.gz
```

Para instalar la base de datos realizar las siguientes acciones:

```
cd db-VERSION
cd build-unix
../dist/configure
make
su
make install
```

Para realizar la instalación (sentencia `make install`) normalmente es necesario tener permisos de superusuario (sentencia `su` para pasar a usuario `root`). Consulte el manual de instalación incluido en la distribución para más información.

9.5.1.2. Capa de transporte segura

Opcionalmente, si se desea establecer comunicaciones seguras mediante una capa de transporte segura (*Transport Layer Security*) debe instalar las librerías TSL. Para ello instale una distribución de *OpenSSL*

Se puede descargar la última versión de este software así como información sobre su instalación en la dirección http://www.openssl.org/.

9.5.1.3. Servicio de autenticación Kerberos

Los clientes y servidores OpenLDAP soportan autenticación basada en *Kerberos*, en concreto el mecanismo de autenticación SASL/GSSAPI. Si desea utilizar este tipo de autenticación podrá encontrar versiones de este software e instrucciones de uso en Heimdal Kerberos (http://www.pdc.kth.se/heimdal/) o en *MIT Kerberos* (http://web.mit.edu/kerberos/www/).

9.5.1.4. Autenticación simple y capa de seguridad

Si desea tener soporte para autenticación simple y capa de seguridad (*Simple Authentication and Security Layer*) es necesario tener instaladas las librerías SASL. Podrá encontrar una instalación de este software en *Cyrus SASL* (http://asg.web.cmu.edu/sasl/sasl-library.html).

9.5.2. Instalación manual del servidor

La instalación manual del servidor nos permitirá configurar totalmente las opciones del servidor de OpenLDAP: funcionalidad, módulos integrados, localización de los archivos de librería, ejecución y configuración, etc. Para instalar manualmente OpenLDAP en un equipo servidor se deben realizar los siguientes pasos:

❶ **Obtener una distribución de OpenLDAP**. Esta puede ser encontrada en la dirección de la página web oficial:

```
http://www.openldap.org/software/download/
```

Es aconsejable utilizar la última versión estable del software.

❷ **Desempaquetar la instalación**. Para poder realizar la instalación inicialmente habrá de desempaquetar la distribución. Para ello situar el paquete sobre un directorio de trabajo y ejecutar el siguiente comando reemplazando VERSION por la versión del archivo descargado:

```
gunzip -c openldap-VERSION.tgz | tar xvfB -
```

Esto desempaquetará la distribución en un directorio llamado `openldap-VERSION`. Cambiar el directorio de trabajo a éste con la orden:

```
cd openldap-VERSION
```

❸ **Revisar la documentación**. Leer atentamente los documentos incluidos en la distribución `COPYRIGHT`, `LICENSE`, `README` e `INSTALL`.

❹ **Configurar la distribución**. Para establecer la configuración de la distribución se debe lanzar el *script* `configure` al que opcionalmente se le pueden pasar un conjunto de opciones. La lista completa de opciones se pude obtener mediante la ejecución de:

```
./configure -help
```

Si no se desea establecer ninguna configuración específica simplemente ejecutar la sentencia.

```
./configure
```

Si se compila usando la base de datos Berkeley (opción por defecto) y se produce un error en la configuración de la base de datos indicar mediante los siguientes variables de entorno la ubicación de las librerías antes de la configuración:

```
export CPPFLAGS="-I/usr/local/BerkeleyDB.X.X/include"
export LDFLAGS="-L/usr/local/BerkeleyDBX.X/lib"
export LD_LIBRARY_PATH="/DB_INSTALL/build_unix/.libs"
./configure
```

Donde `BerkeleyDB.X.X` hace referencia a la carpeta donde se instaló la versión `X.X` de la base de datos y `DB_INSTALL` es la carpeta donde se encuentran los ficheros de instalación de la base de datos (ver sección 8.5.1.1).

Tras la configuración la distribución estará lista para ser compilada.

❺ **Compilar la distribución**. Para compilar solamente es necesario ejecutar dos sentencias, una para crear las dependencias de los módulos y otra para realizar la compilación en si.

```
make depend
make
```

La compilación no debe informar sobre ningún error.

Comprobar la compilación. Para asegurar una correcta compilación puede lanzarse una comprobación de la misma mediante la sentencia:

```
make test
```

Instalar el software. Una vez configurado y compilado la distribución está lista para ser instalada. Para ello normalmente es necesario contar con permisos de *superusuario*.

```
su root -c 'make install'
```

La instalación se realizará por defecto sobre el directorio `/usr/local` a no ser que se especifique otro en la fase de configuración.

9.5.3. Instalación asistida del servidor

Si no deseamos personalizar la instalación del servidor, alternativamente a la instalación manual, se puede realizar una instalación automatizada mediante un paquete de instalación. El paquete de instalación de OpenLDAP es `slapd`. Para instalarlo simplemente ejecutar la siguiente orden y seguir el asistente de instalación que aparece.

```
apt-get install slapd
```

9.5.4. Configuración del servidor LDAP

Cuando el software se haya compilado e instalado, ya puede configurarlo para utilizarlo en su servidor. Toda la configuración en tiempo de ejecución de slapd se realiza mediante el fichero **`slapd.conf`**, que se instala en el directorio que haya especificado en `--prefix` en el guión de configuración, o bien, si no especificó ninguno, en el directorio predeterminado (suele ser `/usr/local/etc/openldap` o `/etc/ldap`). Dentro del directorio de configuración, en el directorio `schema` encuentra el archivo `core.schema` que se incluyen desde el fichero `slapd.conf` y que incluyen, respectivamente, las definiciones de clases de objetos (*objectclasses*) y atributos para la base de datos de segundo plano de LDAP (*backend*).

El fichero `slapd.conf` está compuesto por una serie de opciones globales de configuración que afectan al servidor `slapd` en su conjunto (incluyendo todas las bases de datos de segundo plano o backends), seguido por cero o más definiciones de *backends*, las cuales contienen información específica de cada base de datos.

Las opciones globales de configuración pueden anularse en un *backend* determinado (para opciones que aparecen más de una vez, se usa la última aparición en el fichero de configuración `slapd.conf`). Se ignoran las líneas en blanco y las líneas de comentario que comienzan por el carácter de «#». Si una línea comienza por un espacio en blanco, se considera una continuación de la línea anterior. El formato general del archivo slapd.conf se puede ver en el listado 9.1.

```
# Opciones globales del servidor y de todas las BD
<opciones de configuración globales>

# definición de la BD y opciones de configuración
database <backend tipo 1>
<opciones de configuración específicas del backend tipo 1>

# definición de la segunda BD y opciones de configuración
database <backend tipo 2>
<opciones de configuración específicas del backend tipo 2>

# definiciones subsiguientes BDs y opciones de configuración
...
```

Listado 9.1. Formato general del archivo slapd.conf.

Los argumentos de la línea de configuración están separados por espacios en blanco o tabuladores. Si un argumento contiene espacios en blanco, el argumento debe encerrarse entre comillas dobles. Si un argumento contiene unas dobles comillas o una barra invertida, el carácter ha de ir precedido de una barra invertida.

La distribución de *OpenLDAP* contiene un fichero de configuración de ejemplo que se instalará en el directorio de configuración.

```
#
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include             /usr/local/etc/openldap/schema/core.schema

# Define global ACLs to disable default read access.

# Do not enable referrals until AFTER you have a working
directory
# service AND an understanding of referrals.
#referral     ldap://root.openldap.org

pidfile         /usr/local/var/run/slapd.pid
argsfile        /usr/local/var/run/slapd.args

# Load dynamic backend modules:
# modulepath  /usr/local/libexec/openldap
# moduleload  back_bdb.la
# moduleload  back_ldap.la
# moduleload  back_ldbm.la
# moduleload  back_passwd.la
# moduleload  back_shell.la


########################################################################
#######
# BDB database definitions
########################################################################
#######

database      bdb
suffix          "dc=my-domain,dc=com"
rootdn          "cn=Manager,dc=my-domain,dc=com"
# Cleartext passwords, especially for the rootdn, should
# be avoid.  See slappasswd(8) and slapd.conf(5) for details.
# Use of strong authentication encouraged.
rootpw          secret
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
directory     /usr/local/var/openldap-data
# Indices to maintain
index objectClass    eq
```

Listado 9.2. Ejemplo de archivo slapd.conf.

9.5.4.1. Opciones globales

Las opciones que se describen en esta sección se aplican a todos los *backends*, a menos que se sobrescriban o anulen específicamente en la definición de un *backend* concreto. Los argumentos de opción que han de sustituirse por texto se encierran entre los signos de mayor menor como `<opción>`.

- `access to <algo> [ by <quien> <niveldeacceso> ]+`

Esta opción concede acceso (especificado en `<niveldeacceso>`) a una serie de entradas o atributos (especificados por `<algo>`) para uno o más peticionarios (especificados en `<quien>`). La característica de control de acceso es bastante potente y permite multitud de configuraciones. Algunos ejemplos sencillos de su uso son:

```
access to * by * read
```

Esta directiva de acceso concede acceso de lectura a todos los usuarios. Si aparece en solitario tiene el mismo efecto que la siguiente línea de `defaultaccess`:

```
defaultaccess read
```

El siguiente ejemplo muestra el uso de una expresión regular para seleccionar las entradas por DN en dos directivas de acceso en las que el orden es significativo.

```
access to dn=".*, o=ua, c=es"
by * search
access to dn=".*, c=es"
by * read
```

El acceso en modo lectura se concede a las entradas que están bajo el sub-árbol `"o=ua, c=es"`, al que se permite el acceso en modo lectura. Si su hubiera invertido el orden de las directivas de acceso, la directiva específica de la Universidad de Michigan nunca hubiera coincidido, puesto que todas las entradas de la U. de M. son también entradas de `c=es`.

El próximo ejemplo vuelve a mostrar la importancia del orden, tanto en lo que se refiere a las directivas de acceso como a las cláusulas "by". También muestra el uso de un selector de atributos para conceder acceso a un atributo específico y a varios selectores <quién> .

```
access to dn=".*, o=U of M, c=US" attr=homePhone
by self write
by dn=".*, o=U of M, c=US" search
by domain=.*\.umich\.edu read
by * compare
access to dn=".*, o=U of M, c=US"
by self write
by dn=".*, o=U of M, c=US" search
by * none
```

Este ejemplo se aplica a las entradas en el subárbol `"o=U of M, c=US"`. Para todos los atributos excepto `homePhone`, la propia entrada tiene puede escribirlos, otras entradas de "U of M" pueden buscar por ellas, y nadie más tiene acceso. La entrada tiene permisos de escritura sobre el atributo homePhone, permisos de búsqueda para otras entradas de la "U of M", permisos de lectura para otros clientes que conecten desde algún lugar del dominio umich.edu, y permisos de comparación para el resto del mundo.

A veces es útil permitir que un DN particular se añada o elimine a sí mismo de un atributo. Por ejemplo, si se desea crear un grupo y permitir que los usuarios lo añadan y eliminen de su atributo `member` en su propio DN, puede lograrse con una directiva de acceso como la siguiente:

```
access to attr=member,entry
by dnattr=member selfwrite
```

El selector dnattr `<quién>` nos dice que el acceso se aplica a entradas listadas en el atributo `member`. El selector `selfwrite access` especifica que tales miembros sólo pueden añadir o eliminar del atributo su propio DN y no otros valores. El añadido del atributo `entry` es necesario, ya que se requiere el access de la entrada para acceder a alguno de los atributos de esa entrada.

Observe que la construcción `attr=member` en la cláusula `<qué>` es un atajo para la cláusula "`dn=* attr=member`" (es decir, que coincide con el atributo member en todas las entradas).

- `attribute <nombre> [<nombre2>] { bin | ces | cis | tel | dn }`

Esta opción asocia a una sintaxis con un nombre de atributo. Por defecto se supone que un atributo tiene sintaxis «cis». Se le puede proporcionar a un atributo un nombre alternativo opcional.

- `defaultaccess { none | compare | search | read | write }`

Esta opción especifica el acceso por defecto que se concederá a los solicitantes que no coincidan con ninguna otra línea de acceso (véanse los ejemplos de control de acceso más abajo). Nótese que un nivel de acceso implica en sí también los niveles de acceso inferiores a él. Por ejemplo, el acceso a escritura implica el acceso a lectura, el acceso a búsqueda y el acceso a comparación.

Valor predeterminado: `defaultaccess read`

- `include <nombrefichero>`

Esta opción ordena a slapd que lea información adicional de configuración desde el fichero especificado, antes de continuar con la línea siguiente del fichero actual. El fichero que se especifica ha de seguir el formato normal de configuración de slapd. Utilice esta opción para incluir ficheros que contengan las clases de objetos (`objectclass`) y definiciones de atributo (`attribute definitions`) de su base de datos de segundo plano o backend. El paquete de software de LDAP viene con los ficheros slapd. oc.conf y slapd.at.conf.

Nota: Tenga cuidado al utilizar esta opción. No hay límite mínimo en el número de opciones include anidadas, ni tampoco se hace una detección de anidamiento para el caso de bucles sin fin.

- `loglevel <numeroentero>`

Esta opción especifica el nivel de detalle con el que el sistema debe registrar en un archivo de registro (syslog) las informaciones de depuración y las estadísticas de funcionamiento (en este caso se registran mediante el servicio LOCAL4 de syslogd(8). Para que esta característica esté habilitada, es necesario haber compilado slapd con la opción de compilación `-DLDAP_DEBUG`, excepto para los dos niveles de estadísticas, que están siempre habilitados. Los niveles de registro son acumulativos. Para visualizar qué números corresponden a cada tipo de depuración, ejecute slapd con la opción -? o consulte la tabla de más abajo. Los valores posibles para <numeroentero> se pueden ver en la tabla 9.6.

Ejemplo: `loglevel 255` hará que grandes cantidades de información vayan a un archivo de registro a través de `syslog`.

Valor predeterminado: `loglevel 256`

- `objectclass <nombre>`

Esta opción define las reglas de estructura o esquema para la clase de objetos (`objectclass`) especificada. Se usa junto con la opción `schema-check`. Su sintaxis es:

```
objectclass <nombre>
[ requires <atributos> ]
[ allows <atributos> ]
```

Código	Descripción
1	trazado de llamadas a función
2	manejo de paquetes de depurado
4	depurado de trazado intensivo heavy trace
8	gestión de conexiones
16	mostrar los paquetes enviados y recibidos
32	procesado del filtro de búsqueda
64	procesado de ficheros de configuración
128	procesado de listas de control de acceso
256	estadísticas de registro de conexiones/operaciones/resultados
512	enviar las entradas de registro de estadísticas
1024	imprimir los backends de comunicación con el intérprete de órdenes
2048	imprimir el análisis completo de depuración
65535	activar depuración completa

Tabla 9.6. Niveles de depuración del servidor LDAP

- `referral <url>`

Esta opción especifica la autoridad en la que basarse cuando slapd no pueda hallar una base de datos local para gestionar una petición.

Ejemplo: `referral ldap://ldap.itd.umich.edu`

Esto remitirá las consultas no locales al servidor LDAP de la *Universidad de Michigan*. Algunos clientes LDAP con capacidades inteligentes

podrán redirigir su consulta a dicho servidor, pero tenga en cuanta que la mayoría de esos clientes no gestionarán URLs sencillas de LDAP que contengan una parte de nombre de máquina y, como opción, una parte de nombre distintivo (dn).

- `schemacheck { on | off }`

Esta opción activa (`on`) o desactiva (`off`) la verificación de estructura. Si la verificación de estructura está activada, se comprobarán las entradas que se añadan o modifiquen, con el fin de garantizar que obedecen a las reglas de estructura o esquema que implica la clase de objetos (`objectclass`) a la que pertenecen, tal y como las definen las correspondientes opciones de la clase de objetos correspondiente. Si la verificación de estructura está desactivada (`off`), esta verificación no se realiza.

Valor predeterminado: `schemacheck off`

- `sizelimit <numeroentero>`

Esta opción especifica el número máximo de entradas que hay que devolver de una operación de búsqueda.

Valor predeterminado: `sizelimit 500`

- `srvtab <nombrefichero>`

Esta opción especifica el fichero `srvtab` en el que `slapd` puede encontrar las claves kerberos necesarias para autentificar a los clientes que usen kerberos. Esto opción es significativa únicamente si Usted utiliza autentificación por kerberos, que ha de activarse en el momento de la compilación incluyendo las definiciones apropiadas en el fichero `Make-common`.

Valor predeterminado: `srvtab /etc/srvtab`

- `timelimit <numeroentero>`

Esta opción especifica el número máximo de segundos (en tiempo real) que slapd pasará contestando una petición de búsqueda. Si pasado ese tiempo no se ha contestado una petición, se devolverá un resultado que indicará *exceeding time*, «tiempo sobrepasado».

Valor predeterminado: `timelimit 3600`

9.5.4.2. Opciones generales del backend

Las opciones descritas en este apartado sólo se aplican a las bases de datos o backend en el que se incluyan. Estas opciones están soportadas para todos los tipos de backend.

- `database <tipobasededatos>`

Esta opción marca el comienzo de la definición de una nueva instancia de base de datos. <tipobasededatos> debe ser una de las siguientes: ldbm, shell, o passwd dependiendo del backend sobre el que servirá la base de datos.

Ejemplo: `database ldbm` marca el comienzo de la definición de una nueva instancia de una base de datos con backend LDBM.

- `lastmod { on | off }`

Esta opción controla si slapd mantendrá automáticamente para cada entrada los atributos modifiersName, modifyTimestamp, creatorsName y createTimestamp.

Valor predeterminado: `lastmod off`

- `readonly { on | off }`

Esta opción pone a la base de datos en modo *sólo lectura*. Si se intenta modificar la base de datos devolverá un error.

Valor predeterminado: `readonly off`

- `replica host=<nombredemaquina>[:<puerto>]`

Esta opción especifica una dirección para la duplicación o réplica esta base de datos. Su sintaxis completa es:

```
replica host=<nombredemaquina>[:<puerto>]
"binddn=<DN>"
bindmethod={ simple | kerberos }
[credentials=<contraseña>]
[srvtab=<nombrefichero>]
```

El parámetro `host` especifica en qué máquina (y opcionalmente, en qué puerto) puede encontrarse la instancia del slapd esclavo. para `<nombremáquina>` puede usarse lo mismo un nombre que una dirección IP. Si no se proporciona el parámetro `<puerto>` se usará el puerto estándar de LDAP, el 389.

El parámetro `binddn` proporciona el DN al que se vinculará el slapd esclavo para sus actualizaciones. Ha de tratarse de un DN que tenga acceso de lectura y escritura a la base de datos del esclavo, que normalmente aparece como `rootdn` en el fichero de configuración del esclavo. También tiene que coincidir con la opción updatedn en el fichero de configuración del slapd esclavo. Puesto que los DN son proclives a contener espacios incrustados, la cadena completa `"binddn=<DN>"` ha de estar encerrada entre comillas.

`bindmethod` puede ser o bien simple o bien kerberos, dependiendo de si se usa autentificación sencilla basada en contraseñas o bien kerberos cuando se conecte con el slapd esclavo. La autentificación sencilla precisa que se proporcione una contraseña válida. La autentificación mediante kerberos precisa de un fichero srvtab válido.

El parámetro `credentials`, que sólo se precisa si se usa autentificación sencilla, proporciona la contraseña para `binddn` en el slapd esclavo.

El parámetro `srvtab`, que sólo se precisa si se usa autentificación mediante kerberos, especifica el nombre de fichero que aloja la llave kerberos para el slapd esclavo. Si se omite, se utiliza el fichero `/etc/srvtab`.

- `replogfile <nombrefichero>`

Esta opción especifica el nombre del fichero de registro de duplicación (registro de réplica) en el cual slapd registrará los cambios. El registro de duplicación generalmente lo escribe slapd y lo lee slurpd. Esta opción normalmente sólo tiene efecto si se usa slurpd para duplicar la base de datos. Sin embargo, puede utilizarla también para generar un registro de transacciones, si slurpd no se está ejecutando. En este caso, necesitará truncar periódicamente el fichero, pues de otra manera crecería indefinidamente.

- `rootdn <dn>`

Esta opción identifica al DN de una entrada no sujeta a control de acceso o a restricciones en los permisos de administración para las operaciones en esta base de datos.

Ejemplo: `rootdn "cn=Manager, o=U of M, c=US"`

- rootkrbname <nombrekerberos>

Esta opción especifica un nombre kerberos que funcionará en todos los casos para el DN dado anteriormente, con independencia de que exista una entrada con el DN especificado o de que tenga el atributo krbName. Esta opción es útil al crear una base de datos y también cuando se utilice slurpd para proporcionar servicios de duplicación (servicios de réplica).

Ejemplo: `rootkrbname admin@umich.edu`

- rootpw <password>

Esta opción especifica una contraseña, que funcionará en todos los casos, para el DN dado anteriormente, con independencia de que el DN en cuestión exista o ya tenga contraseña. Esta opción es útil al crear una base de datos y también cuando se utilice slurpd para proporcionar servicios de duplicación (servicios de réplica). Evite tener una contraseña de texto sencillo acompañando a esta opción. Proporcione una contraseña cifrada (puede usar una entrada del fichero de Unix `/etc/passwd/`). slapd soporta también otros métodos de cifrado.

Ejemplos: `rootpw secret rootpw {crypto}contraseña_cifrada`

- suffix <dn sufijo>

Esta opción especifica el sufijo DN de consultas que se le pasará a la base de datos de backend. Pueden proporcionarse múltiples líneas de sufijo, y se requiere al menos una para cada definición de base de datos.

Ejemplo: `suffix "o=University of Michigan, c=US"`

Las consultas que tengan un DN terminado en `"o=University of Michigan, c=US"` se le pasarán a este backend de base de datos.

Nota: cuando se selecciona el backend al que hay que pasarle la consulta, slapd examina la línea o líneas de sufijo en cada definición de base de datos en el orden en que aparecen en el fichero. De esta manera, si el sufijo de una base de datos es el prefijo de otra, dicho sufijo debe aparecer después que el prefijo en el fichero de configuración.

- updatedn <dn>

Esta opción sólo se aplica a un slapd esclavo. Especifica el DN al que se le permite hacer cambios en la duplicación. Generalmente se trata del DN al que slurpd se vincula cuando hace cambios a la duplicación o réplica.

9.5.4.3. Opciones específicas del backend LDBM

Las opciones de esta categoría sólo se aplican a la base de datos de backend LDBM. Es decir, tienen que ir después de una línea `database ldbm` y antes de otra línea de `database`.

- `cachesize <numeroentero>`

Esta opción especifica a instancia de la base de datos de backend LDBM el número de entradas en la memoria caché interna que ha de mantener.

Valor predeterminado: `cachesize 1000`

- `dbcachesize <numeroentero>`

Esta opción especifica el tamaño en bytes de la memoria caché interna asociada con cada fichero de índice abierto. En caso de no estar soportada por el método de base de datos subyacente, esta opción se ignora sin mayores avisos. El incremento de este número utilizará más memoria, pero también causará un aumento espectacular del rendimiento, especialmente durante las modificaciones o a la hora de construir los índices.

Valor predeterminado: `dbcachesize 100000`

- `directory <directorio>`

Esta opción especifica el directorio donde residen los ficheros LDBM que contienen la base de datos y sus ficheros asociados.

Valor predeterminado: `directory usr/tmp/`

- `index {<listadeatributos> | default} [pres,eq,approx,sub,none]`

Esta opción especifica qué índices hay que mantener para un atributo especificado. Si se proporciona únicamente una `<listadeatributos>` se mantendrán todos los índices posibles.

Ejemplos: `index cn index sn,uid eq,sub,approx index default none`

Este ejemplo hará que se dé mantenimiento a todos los índices para el atributo cn: que se mantengan índices de igualdad, subcadenas y cadenas aproximadas en el caso de los atributos sn y uid; y que no se mantengan índices para todos los demás atributos.

- `mode <numeroentero>`

Esta opción especifica qué permisos de ficheros (modo de protección) debe tener el índice de la base de datos recién creada.

Valor predeterminado: `mode 0600`

9.5.5. Puesta en marcha del servidor LDAP

Slapd puede ejecutarse de dos maneras diferentes, como demonio o servicio permanente, o bien desde inetd(8). Se recomienda la ejecución como demonio permanente, sobre todo si usa el backend de LDBM. Ello permitirá al backend beneficiarse del uso de memoria de almacenamiento intermedio (caché) y evita problemas de acceso compartido a los ficheros de índices de LDBM. Si únicamente ejecuta un backend de tipo SHELL o PASSWD, entonces sí puede considerar la opción de ejecutar slapd desde inetd.

Slapd soporta las siguientes opciones de línea de órdenes:

```
-d <nivel> | ?
```

Esta opción fija el nivel de depuración de slapd en `<nivel>`. Cuando el nivel es un carácter `?', se muestran los distintos niveles de depuración y slapd termina, con independencia de cualquier otra opción que se introduzca. Los niveles de depuración pueden verse en la tabla 9.6.

Los niveles de depuración son acumulativos. Si desea trazar llamadas a funciones y observar qué fichero de configuración se está procesando, fije el nivel de depuración al resultado de la suma de estos dos niveles (en este caso, 65).

```
-f <nombrefichero>
```

Esta opción especifica un fichero de configuración alternativo para slapd. Esta opción es muy útil si se desea ejecutar dos instancias del servidor LDAP, cada una con una configuración distinta.

```
-i
```

Esta opción le especifica a slapd que se ejecute desde inetd en vez de hacerlo como un demonio o servicio independiente. En la próxima sección encontrará más detalles sobre la ejecución de slapd desde inetd.

```
-p <puerto>
```

Esta opción especifica un puerto TCP alternativo en el que slapd se mantendrá a la escucha para las conexiones. El puerto por defecto es el 389.

9.5.5.1. Ejecución del servidor como servicio independiente

Como norma general, slapd se ejecuta de la siguiente manera:

```
# $<SLAPD_DIR>/slapd [<opción>]*
```

donde `<SLAPD_DIR>` tiene el valor que le haya asignado en el proceso de configuración previo a la compilación, y `<opción>` es una de las opciones descritas más arriba. A menos que haya especificado un nivel de depuración, slapd se desvinculará automáticamente del terminal desde el que lo lanzó, y se ejecutará en segundo plano, en modo demonio o servicio. Cualquiera de las opciones de más arriba pueden darse en la línea de órdenes para hacer que slapd cargue un fichero de configuración diferente, o que escuche en otro puerto, etcétera.

Véase el siguiente ejemplo de comienzo de slapd:

```
$<SLAPD_DIR>/slapd -f /home/malere/mi_slapd.conf -d 255
```

Para finalizar con seguridad al servidor slapd, debe utilizar una línea de órdenes como la siguiente:

```
kill -TERM <PID SLAPD>
```

donde `<PID SLAPD>` en el PID del servidor slapd. Este puede ser consultado mediante el comando `ps`.

En *RedHat* y distribuciones basadas en rh, como *Mandrake* y *Esware*, se puede utilizar el script `/etc/rc.d/init.d/opeldap` con el argumento `start`, `stop` o `restart` para iniciar, parar o reiniciar el servicio. En *Debian* y distribuciones basadas en ella (*Corel*, *Storm*, *Citius*) el script de inicio es `/etc/init.d/openldap`.

9.5.5.2. Ejecución de slapd desde inetd

Lo primero que hay que tener en cuenta es que no siempre es eficiente ejecutar `slapd` desde `inetd`. Si se está utilizando unbackend LDBM no suele ser una buena idea. Si el servidor tiene activos muchos servicios la sobrecarga que supone ejecutarlo desde inetd también lo convierte en una mala idea. En otros casos podría ser interesante ejecutar `slapd` desde `inetd`. Los pasos necesarios para ellos son los siguientes.

El primer paso es añadir a `/etc/services/` una línea como la siguiente:

```
ldap 389 # ldap directory service
```

El segundo paso es añadir una línea como la siguiente a su fichero `/etc/inetd.conf`:

```
ldap stream tcp nowait nobody <SLAP_DIR>/slapd slapd -i
```

donde `ETCDIR` tiene el valor que le haya asignado en la configuración previa a la compilación. Finalmente envíele a inetd una señal -HUP y ya tendrá su configuración:

```
killall -HUP inetd
```

o

```
kill -HUP <PID DE INET>
```

9.5.6. Herramientas y utilidades

Con la distribución de *OpenLDAP* podemos encontrar dos tipos de utilidades. Por un lado están las denominadas *herramientas de cliente* que nos permiten modificar, borrar, añadir entradas en el servidor LDAP de una forma remota. Para usar dichas herramientas debe estar activo el servidor LDAP. El otro conjunto de herramientas trabajan sobre la base de datos o *backend* directamente por lo que no necesitan que `slapd` haya sido lanzado. Estas son más útiles cuando se pudieran producir inconsistencias en la base de datos porque varias personas estuvieran actuando sobre el servidor LDAP. A continuación se describen las principales herramientas existentes en cada una de los grupos con sus opciones más características. Encontrará más información sobre dichas utilidades en las páginas *man* de cada herramienta.

9.5.6.1. Herramienta de cliente

- **ldapmodify**

Modifica entradas de un directorio LDAP aceptando la introducción de datos a través de un fichero o de la línea de comandos si no se especifica.

Sintaxis: `ldapmodify [-a] [-r] [-n] [-w passwd] [-H ldapuri] [-D binddn] [-p ldapport] [-f file]`

`-a`: Añade nuevas entradas
`-r`: Remplazar los valores existentes
`-n`: simula la operación, pero no realiza el cambio.
`-f`: Lee la entrada del ficheroLDIF especificado
`-H`: Especifica la URI del servidor LDAP
`-D`: utiliza el dn que nos permitirá realizar la operación.

El fichero debe tener como primera línea el `dn` sobre el que se trabaja. A continuación aparecerá el atributo `changetype` con el valor `add`, `delete`, `modify` o `modrdn` según lo que se quiera hacer.

Con `add` a continuación usaríamos la sintaxis de los ficheros LDIF.

Con `modify` a continuación indicaras otro atributo que puede ser `add`, `delete` o `replace` indicando como valor de este atributo sobre que atributo de la entrada quieres hacer la operación. Después colocaremos el atributo sobre el que realizaremos la operación y el nuevo valor a tomar. Si la operación es de borrado no hace falta indicar esta última línea, a no ser que queramos borra solo los atributos con ese valor dado.

Con `delete` borra la entrada con el dn especificado.

Con `modrdn` cambia el `rnd` existente de la entrada por uno nuevo.

```
dn:cn=Alex Garcia Perez, dc=eps,dc=ua,dc=es
changetype:modrdn
newrdn : cn=Alejandro Garcia Perez
```

Ejemplos de `ldapmodify`:

```
dn : cn=Alex Garcia Perez, ou=alumnos, dc=eps,
dc=ua,dc=es
changetype : modify
```

```
replace : sn
sn : Lopez Alegria
-
add : title
titl e: Grand Poobah
-
add: postalCode
postalCode : 02005
-
delete : street
-
```

- **ldapadd**

Añade entradas al directorio aceptando dichos datos a través de un fichero LDIF o de la línea de comando. En realidad se trata de un enlace fijo a `ldapmodify -a`. La sintaxis y opciones son las mismas que `ldapmodify`.

- **ldapsearch**

Busca entradas en el directorio LDAP.

Sintaxis: `ldapsearch [opciones] filtro [atributos]`

Las posibles opciones del comando son:

`-b base`: indica el punto base de la búsqueda.
`-f fichero`: Lee la entrada de búsqueda del fichero especificado
`-H ldapuri`: Especifica la URI del servidor LDAP
`-D dn`: utiliza el dn que nos permitirá realizar la operación.

Con filtro establecemos los patrones que tienen que cumplir los registros a buscar (se permiten los comodines) y en atributos indicamos opcionalmente los atributos que se mostrarán de los registros encontrados.

Ejemplos de uso:

```
ldapsearch -b "dc=eps,dc=ua,dc=es" "objectclass=*"
```

En el primer caso se mostrarán todas las entradas del directorio `eps.ua.es`.

```
ldapsearch -b "dc=eps,dc=ua,dc=es" "objectclass=person" sn
```

En este segundo se mostrará el tributo `sn` de las entradas que sean personas del directorio `eps.ua.es`.

- **ldapdelete**

Elimina entradas del directorio mediante un fichero o desde línea de comando.

-f fichero: Lee la entrada del ficheroLDIF especificado
-H ldapuri: Especifica la URI del servidor LDAP
-D dn: utiliza el dn que nos permitirá realizar la operación.

$ ldapdelete –D "cn=root,dc=eps,dc=ua,dc=es"
> cn=Juan Perez Garcia, ou=alumnos, dc=eps, dc=ua,dc=es

Borraría la entrada para Juan Pérez García

9.5.6.2. Herramientas de base de datos

- **slapadd**

Añade entradas desde un fichero LDIF a una base de datos *SLAPD*. Actúa sobre la base de datos indicada y le añade las entradas descritas en el LDIF. Si no se especifica un fichero LDIF la información será leída de la entrada estándar.

Sintaxis: `slapadd [-l <inputfile>] [-f <slapdconfigfile>] [-d <debuglevel>] [-n <integer>|-b <suffix>]`

`-d`: indica el nivel de depuración.
`-n`: indica que base datos será modificada en función de un número que indica la posición (primera, segunda, tercera,…) en el fichero de configuración.
`-b`: indica que base de datos será modificada en función del sufijo de la misma.

```
_# cat /tmp/top.ldif
## Construye el nodo raíz
dn: dc=dtic,dc=ua,dc=es
dc: dtic
objectclass: dcObject
objectclass: organizationalUnit
ou: Dtic Dot Ua Dot Es

## Construye el ou profesores
dn: ou=profesores,dc=dtic,dc=ua,dc=es
ou: profesores
objectclass: organizationalUnit

_# slapadd -v -l /tmp/top.ldif
added: "dc=dtic,dc=ua,dc=es"
added: "ou=profesores,dc=dtic,dc=ua,dc=es"
```

Listado 9.3. Creación de entradas básicas para dtic.ua.es.

`-f`: especifica un fichero de configuración alternativo. Si no se indica se utiliza el fichero por defecto de `slapd`.

`-l`: especifica el LDIF de donde obtendrá la(s) entrada(s) a insertar.

Ejemplo: `slapadd -l alumnos.ldif`

- **slapcat**

Extrae las entradas de una base de datos LDAP en formato LDIF. Si no se especifica un fichero se muestran por la salida estándar.

Sintaxis: `slapcat -l <filename> [-f <slapdconfigfile>] [-d <debuglevel>] [-n <databasenumber>|-b <suffix>]`

`-d`: indica el nivel de depuración.

`-n`: indica que base datos será modificada en función de un número que indica la posición (primera, segunda, tercera,…) en el fichero de configuración.

`-b`: indica que base de datos será modificada en función del sufijo de la misma.

`-f`: especifica un fichero de configuración alternativo. Si no se indica se utiliza el fichero por defecto de `slapd`.

`-l`: especifica el fichero LDIF donde insertará las entradas extraídas.

Ejemplo: slapcat –l salida.ldif

- **slapindex**

Esta herramienta se utilizará para la regeneración de índices de la base de datos.

Sintaxis: `slapindex [-f <slapdconfigfile>] [-d <debuglevel>] [-n <databasenumber>|-b <suffix>]`

`-d`: indica el nivel de depuración.

`-n`: indica que base datos será modificada en función de un número que indica la posición (primera, segunda, tercera,…) en el fichero de configuración.

`-b`: indica que base de datos será modificada en función del sufijo de la misma..

`-f`: especifica un fichero de configuración alternativo. Si no se indica se utiliza el fichero por defecto de `slapd`.

- **slappasswd**

Genera una contraseña de usuario cifrada para usar con `ldapmodify` o el valor `rootpw` para el fichero de configuración `slapd.conf`.

Sintaxis: `slappasswd [-h schema]`

-h schema: Se especifica uno de los siguientes mecanismos de encriptación. `{CRYPT}`, `{MD5}`, `{SMD5}`, `{SSHA}` y `{SHA}`. Por defecto utiliza `{SSHA}`.

9.5.6.3. Herramientas gráficas

Independientemente de las utilidades que se distribuyen con *OpenLDAP* existen multitud de clientes gráficos para multitud de plataformas que nos permiten acceder al directorio de forma sencilla.

Algunos de ellos son:

- **GQ**: un cliente LDAP gráfico basado en GTK que soporta el cifrado a través de TLS y permite crear y modificar directorios

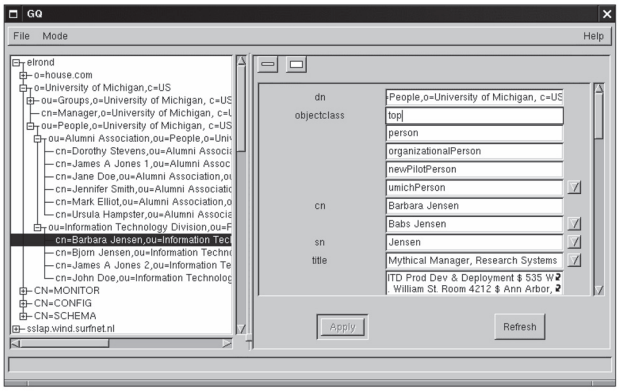


Figura 9.9. Herramienta GQ

- **LDAP Browser/Editor**: Una herramienta basada en Java

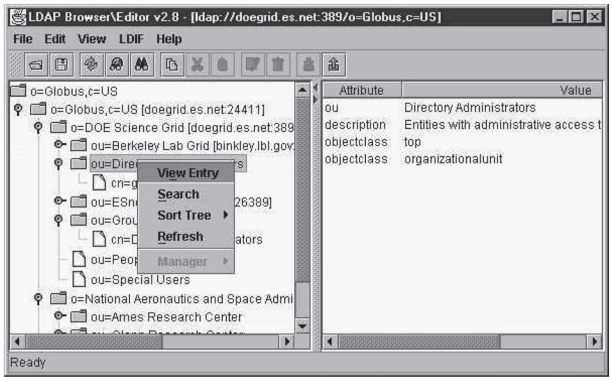


Figura 9.10. LDAP Browser/Editor

9.5.7. Replicación

En algunos casos por requerimientos del sistema o por rendimiento puede ser interesante tener más de un servidor LDAP en producción sirviendo el mismo directorio. Mediante el servidor `slurpd` *OpenLDAP* se

pueden realizar replicación de directorios. Este servidor de replicación (que se ejecuta en el mismo nodo que el *servidor primario*) se nutre del registro de acciones realizado por el servidor primario (`slapd`) y mediante el protocolo LDAP replicar las acciones en otros servidores (*servidores secundarios*) para mantener una copia del directorio. En la figura 9.11 se puede ver un esquema del funcionamiento de *OpenLDAP* con replicación.

Cuando un cliente se conecta al servidor primario para realizar una modificación, el servidor, tras realizarla, anota el cambio en el *log de replicación*. Posteriormente el servidor `slurpd` obtiene la(s) nueva(s) modificación(es) del *log de replicación* y mediante el protocolo LDAP realiza las modificaciones en el(los) servidor(es) secunario(s). En el caso de que el cliente se conecte a un servidor secundario para realizar una modificación, este le devolverá una referencia (referral) al servidor primario y el cliente reenviará la solicitud de modificación a este. Después el servidor primario propagará los cambios como se ha comentado anteriormente.

9.5.7.1. Configuración de la replicación

Para poner en marcha un servicio de directorio replicado hay que realizar, una vez instalados todos los servidores LDAP necesarios, los pasos que se describen a continuación. Normalmente el servidor de replicación `slurp` utiliza el mismo archivo de configuración que `slapd`.

- Configuración del servidor primario

1. Incluir en el archivo de configuración del servidor primario una directiva `replica` por cada servidor secundario que se tenga. El parámetro `binddn` que indica la rama que se replica se tiene que corresponder con la opción `updatedn` de cada servidor secundario.
2. Incluir una directiva `replogfile` que indica el archivo de log que se utilizará.

- Configurar los servidores secundarios

Una vez instalados cada servidor secundario realizar los siguientes pasos:

1. Asegurarse que en el fichero de configuración **no** aparezca ninguna directiva `replica` ni `replogfile`.

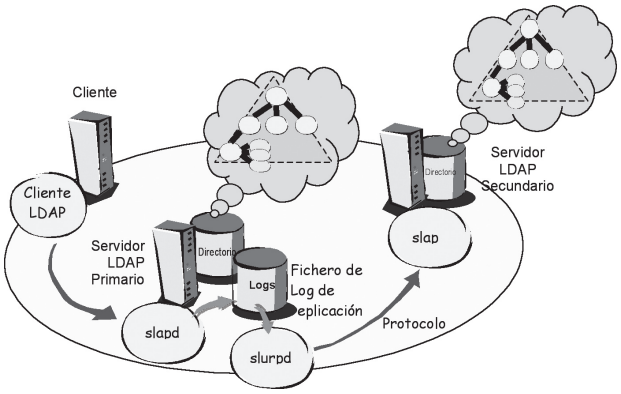


Figura 9.11. Esquema de funcionamiento de LDAP con replicación

2. Incluir una directiva `updatedn` que indica el mismo valor que la opción `binddn` de la directiva `replica` del servidor. Tienen que existir permisos de escritura en este `dn`.

9.5.7.1. Copiar la base de datos

Copie la base de datos del servidor primario a los diferentes servidores secundarios. Antes de ello asegúrese de que todos los servidores están parados. Para copiar la base de datos realice una copia de todos los archivos que componen dicha base. Los directorios donde se sitúan las bases de datos del servidor primario y de los secundarios estarán referenciados en sus respectivos archivos de configuración.

9.5.7.2. Reiniciar servidores OpenLDAP

Reinicie el servidor primario así como todos los servidores secundarios. Para comprobar que se están generando los *logs* pertinentes realice una modificación en el servidor primario y compruebe que esta es anotada en el archivo de *log*.

9.5.7.3. Iniciar el servidor de replica

Inicie el servidor de replicación `slurpd`. El servicio se encuentra normalmente en el mismo directorio que `slapd` y sus argumentos son similares a este (consulte las páginas *man* para más información).

9.6. CASO DE USO: ACTIVE DIRECTORY

El **Directorio Activo** (*Active Directory*) es el directorio usado por *Microsoft Windows 2003* (y anteriormente usado por *Microsoft Windows 2000*) para gestionar los dominios en plataformas Windows. Con *Active Directory* se gestiona y organiza de forma centralizada todos los recursos de la red así como información sobre usuarios y los accesos de éstos a los recursos. Al conjunto de recursos que se gestionan desde el directorio se conoce en las plataformas *Windows* como *dominio*. Se trata de una implementación de LDAP, quizás menos flexible que otras como *OpenLDAP* pero muy integrada con el sistema operativo, que provee un catálogo global para localizar fácilmente los objetos del directorio. Además de con LDAP, Active Directory está intimamente relacionado con otros protocolos como DHCP, para la configuración de red dinámica de clientes, DNS, para la resolución de nombres, SNTP, para la gestión del tiempo en la red, *Kerberos*, para autenticación de usuarios y equipos, y certificados X.509, para la transmisión segura de la información.

En la nomenclatura de *Active Directory* se usa un espacio de nombres igual al de DNS si bien son espacios totalmente diferenciados. Esto significa que si bien un servidor puede tener un nombre en *Active Directory* como n30.dtic.ua.es igual a su nombre DNS, las resoluciones se realizan de forma independiente: cuando necesito saber la IP del nodo se utiliza DNS y cuando se necesita localizar el nodo en el directorio se utiliza *Active Directory*.

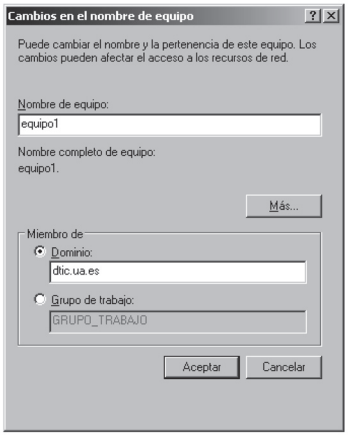


Figura 9.12. Inserción de un cliente en un dominio

9.6.1. Estructura lógica

En función de la distribución de la organización que estamos representando con el directoriosse establece una estructura lógica del mismo que facilita la administración y el uso del directorio. Para ello contamos con una serie de unidades organizativas como son el dominio, árbol y bosque que nos facilitan su estructuración.

9.6.1.1. Dominio

El dominio es la unidad base del directorio. Un dominio lo conforman el conjunto de nodos y usuarios que comparten un directorio común. Como veremos más adelante un Directorio Activo podrá estar compuesto por más de un dominio. Cada dominio estará controlado por al menos un equipo que actúa de **controlador de dominio**. Cada dominio se identifica por un sufijo DNS que será el sufijo común de todos los nodos del dominio.

El uso de dominios nos permite realizar las siguientes opciones en su ámbito de gestión:

- Gestión de usuarios que tendrá acceso al dominio
- Gestión de la seguridad, mediante directivas de seguridad y listas de control de acceso (ACL)
- Replicación de la información, permitiendo mantener copias del dominio en otros controladores de dominio.
- Uso de políticas de grupo.
- Delegación de permisos

9.6.1.2. Árboles y Bosques

En muchos casos, por causas de organización administrativa, funcional o geográfica, se pueden establecer más de un dominio en una misma organización.

Cuando los dominios comparten un mismo espacio de nombres, es decir que tienen un sufijo común entre ellos, y que por lo tanto se puede establecer una relación jerárquica entre ellos, estos conforman una unidad de agrupación conocida como *árbol*. En un árbol existen relaciones predeterminadas entre un padre y sus hijos.

Cuando tenemos diferentes dominios que no comparten el mismo espacio de nombres (no tienen un sufijo DNS común) estos se distribuyen

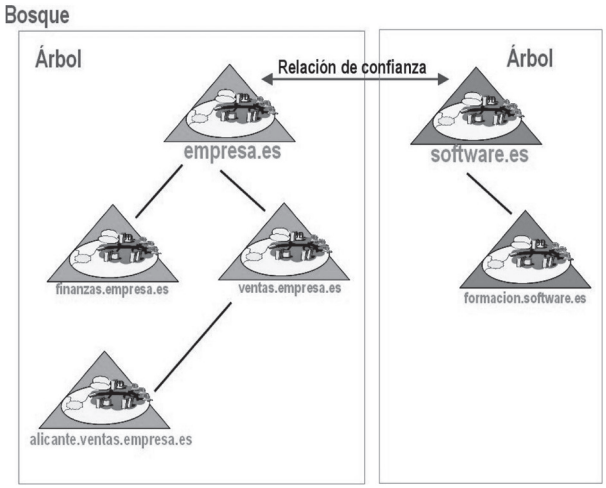


Figura 9.13. Esquema de un bosque en Active Directory

formando un *bosque*. Los bosques están compuestos por diversos árboles relacionados entré si por medio de *relaciones de confianza*. Todos los árboles en un bosque comparten un esquema común, configuración, y un *Catalogo Global*.

9.6.1.3. Unidades Organizativas

Mediante las *unidades organizativas* (OU) el administrador del Directorio Activo pude agrupar los diferentes objetos que lo componen. Una unidad organizativa hace de contenedor de objetos o de otras unidades organizativas de la misma forma que en un sistema de archivos un directorio contiene un conjunto de archivos o de otros directorios. De esta forma se crea un árbol organizativo dentro de cada dominio donde se pueden estructurar todos los objetos como usuarios, grupos o recursos según las necesidades de la organización.

Con una unidad organizativa se simplifica la administración ya que podemos aplicar propiedades a toda la unidad y esta será trasladada a todos los objetos contenidos en ella. Con las unidades organizativas también se puede realizar fácilmente una delegación administrativa por zonas.

En todo Directorio Activo existe un único *dominio raíz de bosque* que es el domino raíz del primer árbol creado.

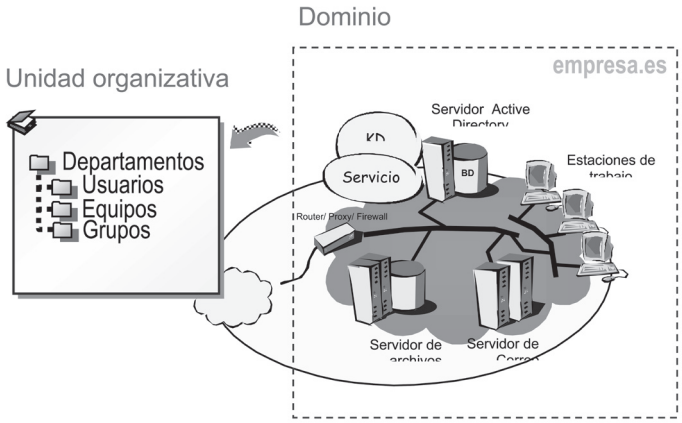


Figura 9.14. Unidades organizativas de un dominio

9.6.1.4. Relaciones de confianza

Los diferentes dominios se relacionan entre sí mediante relaciones de confianza. Las relaciones permiten establecer si el controlador de un dominio puede tener acceso a los objetos de otro. Las relaciones pueden ser creadas por el administrador del dominio o pueden crearse de forma automática en algunos casos. Las relaciones de confianza pueden ser bidi-

Nombre	Automática	Bidireccional	Transitiva	Descripción
Raíz de árbol	Sí	Sí	Sí	Relación creada entre los diferentes dominios raíz de los árboles de un mismo bosque
Principal-secundario	Sí	Sí	Sí	Relación entre un dominio y sus subdominios del árbol
De acceso directo	No	No	Sí	Mejora el rendimiento en el acceso a recursos desde un dominio a otro
Externa	No	No	No	Permite el acceso a otros bosques o dominios en Windows NT
De bosque	No	No	Sí	Relación entre dominios raíz de dos bosques distintos
De territorio	No	No	Sí o No	Relación con un territorio Kerberos que no sea Windows

Tabla 9.7. Tipos de relación en un Directorio Activo

reccionales, cuando la relación es en ambos sentidos, y transitivas, cuando un domino A está relacionado con otro B y este con otro C implica que A está relacionado con C.

En la tabla 9.7 se muestras los diferentes tipos de relaciones posibles en un *Directorio Activo* con Windows 2003.

9.6.2. Instalación del Directorio Activo

Para instalar un dominio hay que poner en marcha un controlador de dominio en un servidor Windows 2000 o 2003. Para ello contamos con una herramienta visual basada en asistentes llamada **`dcpromo`**. Para lanzarla hay que ir a `Inicio>Ejecutar…` escribir `dcpromo` y pulsar aceptar.

El asistente nos irá guiando en todos los pasos necesarios para crear un dominio raíz, un dominio dentro de un árbol, un nuevo bosque, etc.

Una vez instalado el controlador de dominio en la sección de herramientas administrativas del servidor aparecerán nuevas herramientas como se puede observar en la figura 9.15.

Estas herramientas son:

- **Dominios y confianzas de *Active Directory*** para la gestión de relaciones de confianzas entre dominios.
- **Sitios y servicios de *Active Directory***
- **Usuarios y equipos de *Active Directory***

El uso de estas herramientas es bastante intuitivo mediante el uso de los menús contextuales en los diferentes objetos que muestran en las ventanas. Como ejemplo en la figura 9.16 se ilustra la creación de un nuevo usuario desde la herramienta de *Usuarios y equipos de Active Directory*.

Para realizar una delegación dentro del dominio existe una opción específica en los menús contextuales. Por ejemplo como se ilustra en la figura 9.XXX dentro de la herramienta *Sitios y servicios de Active Directory* podemos delegar el control de un sitio a otro usuario del directorio.

9.6.3. Recursos Compartidos

Como en el resto de redes Windows dentro de un dominio un equipo puede compartir recursos como carpetas o impresoras de forma que otros miembros del dominio puedan hacer uso de dicho recurso. Para ello se utiliza los permisos asignados al recurso para un usuario determinado del

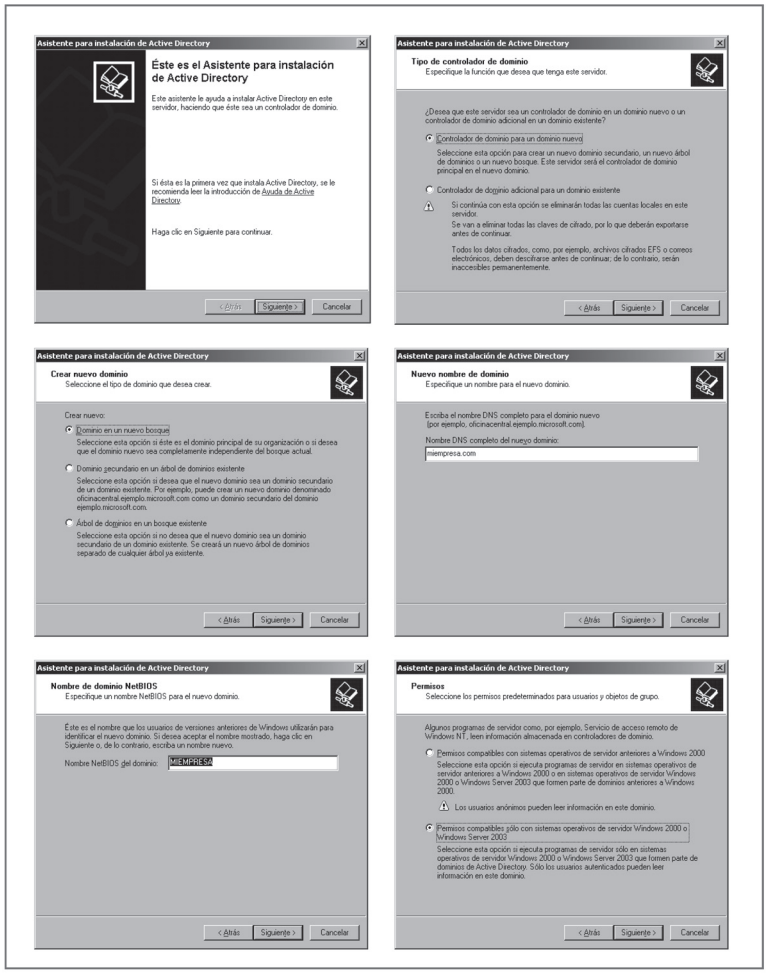


Figura 9.15. Secuencia de pantallas del asistente dcpromo para la creación de un Controlador de Dominio raíz.

dominio. En Windows cuando compartimos una carpeta se asigna los diferentes permisos (como de lectura o escritura) para cada usuario o grupo del dominio. Un dato a tener en cuenta es que cuando la carpeta se encuentra en un sistema de archivos NTFS ésta también tendrá asignados unos permisos para cada usuario. En estos casos para que un usuario del dominio tenga acceso a una carpeta tendrá que tener permisos tanto del sistema de archivos como de carpeta compartida.

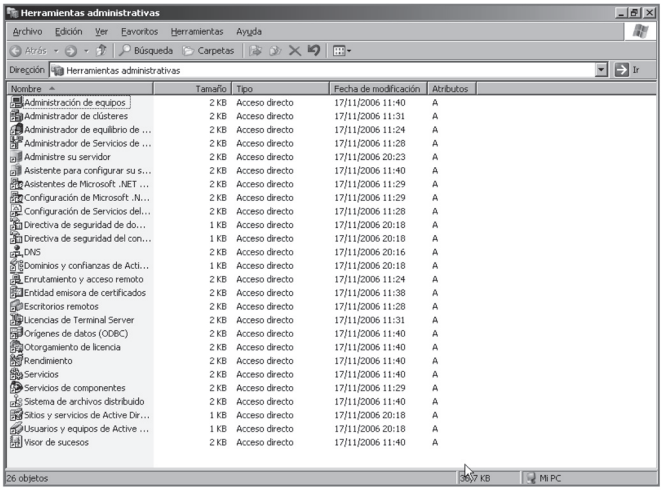


Figura 9.16. Secuencia de pantallas del asistente dcpromo para la creación de un Controlador de Dominio raíz.

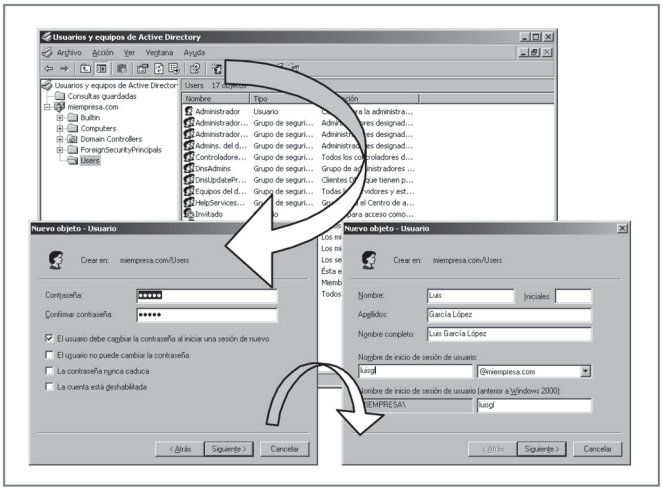


Figura 9.17. Creación de un usuario en Active Directory.

Para compartir una carpeta tendremos que seleccionar “Compartir y seguridad…” desde el menú contextual que aparece al pulsar con el botón secundario sobre la carpeta.

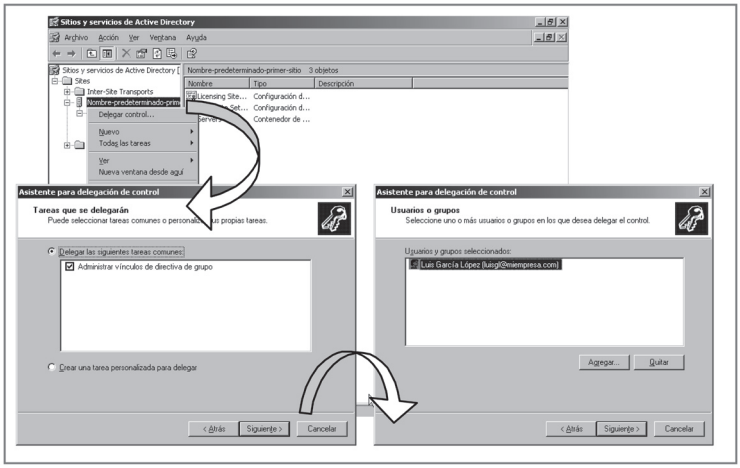


Figura 9.18. Delegación del control de un sitio

Posteriormente para acceder al recurso compartido desde otro equipo utilizaremos la siguiente dirección indicando el nombre del equipo que lo comparte y del recurso:

```
\\equipo\nombre_recuros
```

Otra opción es utilizar una herramienta basada en texto `net`. Para ello abrir una consola de texto (`Inicio->Ejecutar…->cmd`) e introducir las siguientes ordenes:

- Para mostrar los recursos compartidos:

```
net share
```

- Para mostrar información sobre un recurso:

```
net share <nombre>
```

- Para compartir una carpeta situada en `<unidad:path>`:

```
net share <nombre=unidad:path> [{/users:numero|/unlimited}] [/remark:<descripcion>]
```

- Para dejar de compartir un recurso:

```
net share <nombre> /delete
```

- Para mostrar los recursos conectados:

```
net use
```

- Para conectar un recurso a un dispositivo (p.e. `E:`):

```
net use <dispositivo> <\\equipo\nombre> [/user:<usuario>]
```

- Para desconectar un recurso:

```
net use <dispositivo> /delete
```

10. ACCESO A SERVICIOS MEDIANTE LDAP

Durante este capítulo se analizará la incorporación de LDAP como sistema de gestión de usuarios y recursos para diferentes servicios como el acceso a sistemas o servicios web.

10.1. ACCESO DE SISTEMAS LINUX A DOMINIOS LINUX

Por defecto en un equipo Linux la autenticación en la entrada al sistema (el proceso conocido como *login*) se realiza mediante el uso de un nombre de usuario y contraseña que se valida de forma local utilizando normalmente el archivo de configuración `/etc/passwd` donde se almacena la información de los usuarios: identificadores de usuario y grupo (UID y GID), interprete de comandos a usar, directorio *home* del usuario, etc.

Cuando tenemos más de un equipo unidos mediante una red de comunicaciones la opción más inmediata para que los usuarios compartan equipos y recursos es replicar esta información. Esto se puede realizar de forma manual o mediante algún protocolo estándar como NIS.

Otra opción, cada vez más en auge, es utilizar un servidor LDAP para almacenar esa información y realizar una autenticación centralizada al estilo de un dominio en un *Directorio Activo* de *Windows*. Para ello necesitamos evidentemente instalar un servidor LDAP (como podría ser *OpenLDAP*) en un equipo que hará las veces de controlador de dominio como en el caso de *Active Directory*.

Para implementar este sistema tendremos que instalar y configurar un servidor OpenLDAP en un equipo servidor y posteriormente configurar

los clientes para realicen una validación remota a este servidor en vez de la clásica validación local mediante el archivo `/etc/passwd`.

10.1.1. Configuración del servidor LDAP

El primer paso consiste en poner en marcha un servidor LDAP que almacene la información referente a los usuarios y grupos de usuarios de la red o dominio. Inicialmente tendremos que instalar la distribución de *OpenLDAP* como se vio en el capítulo 9.

Seguidamente tendremos que configurar el servidor `slapd` para que actúe de controlador introduciendo o modificando los siguientes valores en su archivo de configuración `slapd.conf` (situado en `/etc/openldap` o donde se decidiera en su instalación):

- Establecer el sufijo del directorio (por ejemplo *miempresa.com*) con la línea:

```
suffix "dc=miempresa, dc=com"
```

- Establecer la cuenta de administrador de la base de datos y su contraseña. Esta cuenta no tiene por que ser una cuenta de acceso nuestro dominio, sólo tiene valor dentro de la administración del directorio.

```
rootdn "cn=admin, dc=miempresa, dc=com"
rootpw <CONTRASEÑA>
```

Si queremos cifrar la contraseña para que no se muestre directamente en el archivo de configuración podremos realizarlo mediante el comando `slappasswd -h {MD5}` e indicarlo en el archivo con el prefijo `{MD5}`:

```
rootpw {MD5}<CONTRASEÑA_CIFRADA>
```

Una vez configurado es conveniente establecer el arranque automático del servicio *OpenLDAP* en el equipo para no tener que realizarlo manualmente cada vez que iniciemos el servidor.

10.1.2. Configuración de los clientes LDAP

La configuración de los clientes LDAP se realiza mediante dos ficheros de configuración: `/etc/openldap/ldap.conf`, fichero propio de OpenLDAP y `/etc/ldap.conf` donde se incluyen opciones de autentica-

ción y gestión de contraseñas. Para establecer la configuración habrá que incluir en el primero de ellos las siguientes sentencias:

```
BASE dc=miempresa,dc=com
HOST server1.miempresa.com
```

Donde `server1.miempresa.com` es el nombre del servidor *OpenLDAP*. Para que cada cliente pueda resolver el nombre del servidor tendrá que existir un servidor DNS que lo resuelva o tendrá que estar incluido en el fichero `/etc/hosts` de dada cliente.

10.1.3. Migración de la información

Para migrar la información existente en los archivos /etc/passwd y /etc/group podemos usar unas herramientas de migración incluidas en la carpeta `migration` de la distribución de LDAP.

Lo primero es indicar la localización y el sufijo del servidor LDAP editando el fichero `migrate_common.ph` e indicando:

```
$DEFAULT_MAIL_DOMAIN = "miempresa.com";
$DEFAULT_BASE = "dc=miempresa,dc=com";
```

después realizaremos los siguientes pasos desde dicha carpeta para incorporar la base del dominio:

```
./migrate_base.pl > base.ldif
ldapadd -x -c -D "cn=admin,dc=miempresa,dc=com" -W -f base.ldif
```

Seguidamente incorporamos los usuarios creando primeramente un fichero LDIF con los usuarios:

```
./migrate_passwd.pl /etc/passwd > usrs.ldif
```

Después se edita el archivo `usuarios.ldif` y se elimina la entrada de `root` ya que por motivos de seguridad no se recomienda su inclusión en el directorio, dejándolo como un usuario local en cada equipo. Posteriormente se integra en el directorio mediante:

```
ldapadd -x -c -D "cn=admin,dc=miempresa,dc=com" -W -f usrs.ldif
```

Por último migramos también los grupos eliminando entre los dos siguientes comandos los grupos de `root` en el fichero `grp.ldif`:

```
./migrate_group.pl /etc/group > grp.ldif
ldapadd -x -c -D "cn=admin,dc=miempresa,dc=com" -W -f grp.ldif
```

A partir de este momento debemos añadir los usuarios y grupos a nuestro sistema añadiéndolos al directorio y dejar los ficheros de usuarios y grupos vacíos a excepción de algunos especiales como `root`. Para saber el formato de las entradas añadir consultar los archivos LDIF temporales que se han creado en la migración.

Otra opción es utilizar una herramienta gráfica que nos facilite el mantenimiento del directorio como es el caso de *Directory Administrador* que se puede encontrar en `http://diradmin.open-it.org`.

10.1.4. Configuración de la autenticación de los clientes

Para que los clientes se autentiquen mediante el servidor LDAP en vez del utilizando los ficheros locales es necesario modificar el proceso de validación (*login*). Este proceso es configurable mediante el uso de dos bibliotecas: PAM (*Pluggable Authentication Module*) que permite a una aplicación (como es el caso de *login*) validar un usuario y una contraseña, y NSS (*Name Service Switch*) que permite obtener datos como el UID y GID de un usuario, su directorio *home*, su interprete de comandos, etc.

Para poder utilizar las librerías PAM y NSS tenemos previamente que instalarlas en los equipos mediante sus respectivos paquetes de instalación.

Para instalar las librerías PAM de acceso a LDAP ejecutar la sentencia:

```
apt-get install libpam-ldap
```

Durante el proceso de instalación se mostrara un asistente (ver figura 10.1) que nos ayudará a configurar el acceso al directorio. Si ya tenemos instalada la librería y queremos volver a configurar estos parámetros utilizar la sentencia:

```
dpkg-reconfigure libpam-ldap
```

De igual forma tenemos que instalar la librería de NSS con la sentencia:

```
apt-get install libnss-ldap
```

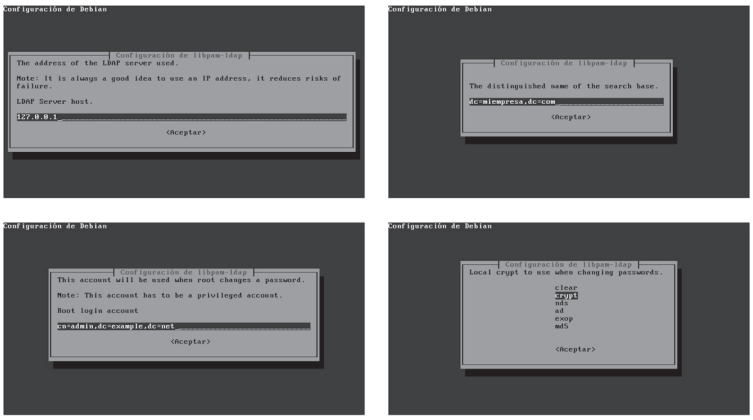


Figura 10.1. Configuración del paquete libpam-ldap.

Con esta instalación también tendremos un asistente de configuración (ver figura 10.XXX) que podremos volver a iniciar mediante la sentencia:

```
dpkg-reconfigure libnss-ldap
```

Figura 10.2. Configuración del paquete libpam-ldap.

Como se puede observar en la última pantalla del asistente hay que configurar el archivo `/etc/nsswitch.conf` para indicar que la autentica-

ción se realizará mediante LDAP. En el archivo `usr/share/doc/libnss-ldap/examples/nsswitch.ldap` se encuentra un ejemplo de los datos que hay que indicar (ver figura 10.XXX).

La configuración realizada se almacena en el archivo de configuración PAM: `/etc/pam_ldap.conf`.

```
passwd:              files ldap
group:       files ldap

hosts:       dns ldap

services:   ldap [NOTFOUND=return] files
networks:   ldap [NOTFOUND=return] files
protocols:  ldap [NOTFOUND=return] files
rpc:        ldap [NOTFOUND=return] files
ethers:     ldap [NOTFOUND=return] files

netmasks:   files
bootparams: files
publickey:  files
automount:  files

aliases:    files
sendmailvars:   files

netgroup:   ldap [NOTFOUND=return] files
```

Listado 10.1. Ejemplo de archivo `/etc/nsswitch.conf` con conexión a LDAP.

Para configurar estas bibliotecas en distribuciones de Red Hat también tenemos la herramienta `authconfig` que mediante un asistente permite configurar los datos de conexión a LDAP.

10.2. ACCESO DE SISTEMAS LINUX A DOMINIOS WINDOWS

Si tenemos un conjunto híbrido de equipos Windows y Linux en red y queremos que estos coexistan entre si dentro de un dominio gestionado por un servidor Windows tendremos que modificar el esquema de *Active Directory* de forma que cumpla también los requerimientos exigidos por un cliente Linux. Esto se debe a que lo clientes Linux en el proceso de

inicio necesitan datos complementarios como su directorio *home*, su interprete de comandos o sus códigos UID y GID.

Para modificar el esquema de *Active Directory* lo más cómodo es usar una utilidad de libre distribución llamada AD4Unix que puede ser encontrada en:

```
http://sourceforge.net/projects/ad4unix/
```

Esta utilidad también incluye una pestaña más en las ventanas de configuración de usuarios y grupos para la gestión de los valores UID, GID, directorio *home*, etc.

10.3. ACCESO DE SISTEMAS WINDOWS A DOMINIOS LINUX

Si tenemos un conjunto híbrido de equipos Windows y Linux en red y queremos que estos coexistan entre si dentro de un dominio gestionado por un servidor Linux tendremos que hacer que este servidor actúe como un controlador de dominio Windows de forma que los equipos clientes Windows se conecten a este de la misma forma que si el servidor fuera un controlador de dominio Windows.

Para ello el directorio del servidor Linux (en nuestro caso un directorio OpenLDAP) tendrá una estructura que partirá de un nodo inicial como por ejemplo `dc=miempresa,dc=com` y a partir de este una estructura especial con usuarios (`ou=Users,dc=miempresa,dc=com`), grupos (`ou=Groups,dc=miempresa,dc=com`) y equipos (`ou=Computers,dc=miempresa,dc=com`).

Para que podamos crear objetos del estilo *Active Directory* en directorio Linux tendremos que configurar el servidor `slapd` incluyendo en su fichero de configuración los *schemas* pertinentes añadiendo la línea:

```
include /usr/local/etc/openldap/schema/samba.schema
```

el esquema de `samba.schema` se puede encontrar en la distribución de Samba.

10.3.1. Configuración de Samba

Con las instrucciones que se han dado hasta ahora se podemos hacer que un cliente Windows se valide contra un servidor LDAP sobre Linux.

Si además queremos compartir recursos como carpetas compartidas del servidores Linux para ser accedidas desde equipos Windows necesitamos configurar el servidor de Samba para que obtenga los datos de usuario del servidor LDAP.

10.4. SERVICIOS WEB

En esta sección se tratara la integración de LDAP como método de autenticación de diferentes servicios Web

10.4.1. Servidor Apache2

Como se vio en el tema 1 el servidor Web Apache 2 permite crear zonas en nuestro sitio que requieren autenticación para su acceso. En el caso de uso expuesto se creaba un fichero de usuarios y contraseñas que Apache utilizaba para validar el acceso a un determinado directorio.

Para grandes cantidades de usuarios o cuando la lista de estos es muy variable es más eficiente plantear la autenticación de estos usuarios mediante un servicio de base de datos o un servicio de directorio. En el caso de Apache existe un módulo que permite establecer una conexión con un servidor LDAP como puede ser *OpenLDAP* para obtener los datos de usuario.

El módulo de Apache se denomina `ldap_module` y pude ser obtenido de la página oficial de apache (`www.apache.org`). Para ello en el proceso de configuración de la distribución usar el argumento `--with-ldap` al lanzar el script `configure` (ver capítulo 1.4.3).

Para configurar el acceso a un directorio mediante una validación con LDAP establecer en el fichero de configuración del sitio las opciones como se muestran en el listado 10.2.

El módulo de LDAP de Apache utiliza un sistema de caché que evita la saturación del servidor LDAP. La caché utilizada es de dos niveles: una para el proceso de conexión y búsqueda y otra para el proceso de comparación. Las opciones que se muestran en la parte superior del listado especifican los valores de configuración de dicha caché:

- `LDAPSharedCacheSize`: Bytes utilizados en la caché.
- `LDAPCacheEntries`: Número de entradas de la caché.
- `LDAPCacheTTL`: Tiempo en segundos que una entrada permanece en la caché de conexión y búsqueda.

```
LDAPSharedCacheSize 200000
LDAPCacheEntries 1024
LDAPCacheTTL 600
LDAPOpCacheEntries 1024
LDAPOpCacheTTL 600

<Location /var/www/privado>

        SetHandler ldap-status
        Order deny,allow
        Deny from all
        Allow from midominio.ejemplo.com
        AuthLDAPURL ldap://127.0.0.1/dc=ejemplo,dc=com?uid?one
        require valid-user

</Location>
```

Listado 10.2. Ejemplo de autenticación en Apache mediante LDAP

- `LDAPOpCacheEntries`: Especifica el número de entradas en la cache de comparación.
- `LDAPOpCacheTTL`: Tiempo en segundos que una entrada permanece en la caché de comparación.

Con la entrada `SetHandler ldap-status` el administrador puede posteriormente monitorizar la caché en función del nombre que se le da desde la dirección `http://servername/cache-info`.

Con la directiva `AuthLDAPURL` establecemos la conexión con el servidor de LDAP. A continuación le indicamos la URL con la siguiente sintaxis:

```
ldap://host:puerto/dnbase?atributo?ambito?filtro
```

donde:

- `host:puerto` es el nombre (o IP) y puerto del servidor LDAP.
- `dnbase`: es la rama del directorio donde se encuentra la información de usuarios.
- `atributo` es el atributo que se tiene que buscar. Por defecto se utiliza `uid`.
- `ámbito` indica el ámbito de la búsqueda. Puede ser `base` o `sub` (por defecto).
- `filtro` es un filtro opcional para la búsqueda.

10.4.2. Servidor IIS

La integración del servidor web de Microsoft IIS con un directorio para la validación es mucho más sencilla e intuitiva que en el caso de Apache y LDAP. En el caso de IIS la integración con los usuarios del Directorio Activo se realiza de forma visual mediante la herramienta de configuración de IIS.

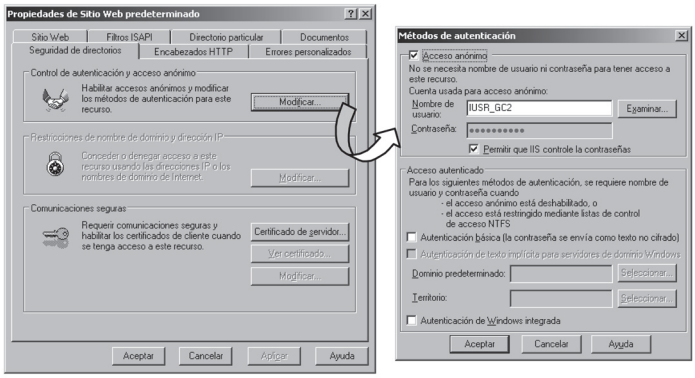


Figura 10.3. Configuración de la autenticación en IIS.

Para ello hay que entrar en las propiedades del sitio o de la carpeta donde queremos establecer un acceso controlado por usuario e ir a la pestaña *Seguridad de directorios*. Allí pulsaremos el botón *Modificar* y estableceremos el dominio y opcionalmente el territorio donde se encuentran los usuarios que van ha ser utilizados para realizar la validación.

10.5. CORREO ELECTRÓNICO

Al igual que en el caso de los servidores web algunos servidores de correo pueden utilizar la información almacenada en un directorio para obtener datos referentes a usuarios y cuentas.

En el caso de *qmail* que se estudió en el capitulo 6 dado que la configuración se realiza utilizando una los directorios *home* de los usuarios una posible incorporación de LDAP vendría por el uso de un directorio actuando como dominio del sistema, tal cual se vio al principio de este capítulo.

Actualmente se está trabajando en una versión de *qmail* con soporte para LDAP llevada a cabo como un parche de los fuentes de *qmail*. Para más información consultar la página web:

```
http://www.qmail-ldap.org
```

En plataformas Windows con un servidor de correo como *Microsoft Exchange* tenemos una gran integración con Active Directory de forma que la gestión de los usuarios está muy relacionada con la de sus cuentas. Por motivos de extensión este tema no será tratado en este libro. Para más información consultar la página web:

```
http://www.microsoft.com/spain/exchange
```